Exhibit A

UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
WACO DIVISION

Webroot, Inc. and Open Text, Inc., ***Plaintiffs,*** **v.** **Sophos Ltd.,** ***Defendant.***	**Civil Action No. 6:22-cv-00240-ADA** **JURY TRIAL DEMANDED**
Sophos Ltd., ***Counterclaim-Plaintiff,*** **v.** **Webroot, Inc., Open Text, Inc., and Open Text Corp.,** ***Counterclaim-Defendants.***	

THIRD AMENDED COMPLAINT FOR PATENT INFRINGEMENT

Pursuant to Federal Rule of Civil Procedure 15(a) (1) and United States District Court for the Western District of Texas Local Rule CV-15, Plaintiffs Webroot, Inc. ("Webroot") and Open Text, Inc. ("Open Text") (collectively "Plaintiffs") hereby submit their Third Amended Complaint for Patent Infringement against Defendant Sophos Ltd. ("Sophos" or "Defendant") and allege the following:

1. This case involves patented technologies that helped to revolutionize, and have become widely adopted in, the fields of malware detection, network security, and endpoint protection. Endpoint protection involves securing endpoints or entry points of end-user devices

(*e.g.*, desktops, laptops, mobile devices, etc.) on a network or in a cloud from cybersecurity threats, like malware.

2. Before Plaintiffs' patented technologies, security platforms typically relied on signatures (*i.e.*, unique identifiers) of computer objects (*e.g.*, computer programs) that were analyzed and identified as "bad" by teams of threat researchers. This approach required antivirus companies to employ hundreds to thousands of threat analysts to review individual programs and determine if they posed a threat.

3. The "bad" programs identified by researchers were compiled into a library and uploaded to an antivirus software program installed on each endpoint device. To detect threats, a resource intensive "virus scan" of each endpoint device was conducted. These virus scans could take hours to complete and substantially impact productivity and performance.

4. Despite substantial investments in resources and time, the conventional systems still were unable to identify and prevent emerging ("zero-day") threats from new or unknown malware. New threats persisted and were free to wreak havoc until a team of threat analysts could identify each one and upload these newly identified threats to an update of the "bad" program library. The updated "bad" program library, including signatures to identify new threats as well as old, then had to be disseminated to all of the endpoint computers, which required time and resource consuming downloads of the entire signature library to every computer each time an update was provided.

5. By the early-to-mid 2000s, new threats escalated as network connectivity became widespread, and programs that mutate slightly with each new copy (polymorphic programs) appeared. These events, and others, rendered the traditional signature-based virus scan systems ineffective for these modern environments.

6. Plaintiffs' patented technology helped transform the way malware detection and network security is conducted, reducing and often even eliminating the shortcomings that plagued signature-based security products that relied on human analysts.

7. Instead of relying on human analysts, Plaintiffs' patented technology enabled the automatic and real-time analysis, identification, and neutralization of previously unknown threats, including new and emerging malware, as well as advanced polymorphic programs.

8. For example, Plaintiffs' patented technology uses information about the computer objects being executed—including, for example, information about the object's behavior and information collected from across a network—along with machine learning technology and novel system architectures, to provide security systems that are effective in identifying and blocking new security threats in real-time in real-world, commercial systems.

9. Plaintiffs' patented technology further includes new methods of "on execution" malware analysis; new architectures that efficiently and effectively distribute workloads across the network; new forensic techniques that enable fast, efficient, and accurate analysis of malware attacks; and new advanced memory scanning techniques.

10. Plaintiffs' patented technology makes security software, platforms, and appliances better at detecting malware by, for example, reducing false positives/negatives and enabling the identification and mitigation of new and emerging threats in near real-time. These improvements are accomplished while at the same time reducing the resource demands on the endpoint computers (*e.g.*, not requiring downloading and using full signature databases and time-consuming virus scans).

11. Plaintiff Webroot has implemented this technology in its security products like Webroot SecureAnywhere AntiVirus, which identifies and neutralizes unknown and undesirable

computer objects in the wild in real-time.

12. Over the years, Plaintiff Webroot has also received numerous accolades and awards for its products and services. For example, Webroot has received 22 PC Magazine Editor's Choice Awards, including "Best AntiVirus and Security Suite 2021." That same year, Webroot also received the Expert Insights Best-of-Endpoint Security award.

13. Plaintiffs currently own more than 70 patents describing and claiming these and other innovations, including U.S. Patent No. 8,418,250 (the "'250 Patent"), U.S. Patent No. 8,726,389 (the "'389 Patent"), U.S. Patent No. 9,578,045 (the "'045 Patent"), U.S. Patent No. 10,257,224 (the "'224 Patent"), U.S. Patent No. 10,284,591 (the "'591 Patent"), U.S. Patent No. 10,599,844 (the "'844 Patent"), U.S. Patent No. 9,413,721 (the "'721 Patent"), 11,409,869 (the "'869 Patent"), 8,181,244 (the "'244 Patent"), 8,201,243 (the "'243 Patent"), 8,856,505 (the "'505 Patent"), 8,719,932 ("the '932 Patent"), and U.S. Patent No. 8,763,123 (the "'123 Patent") (Exhibits 1-13).

14. Plaintiffs' patented technology represents such a vast improvement on the traditional malware detection and network security systems that it has become a widely adopted and accepted approach to providing endpoint security in real-time.

15. Defendant Sophos Ltd (collectively, "Sophos") is a direct competitor of Plaintiffs and provides security software and systems that, without authorization, implement Plaintiffs' patented technologies. Sophos's infringing security software includes, but is not limited to, Intercept X Advanced with EDR and XDR, Sophos Web Appliance, Sophos XG Firewall, and Sophos Synchronized Security, (collectively, "Sophos Security Suite" or "Accused Products").

16. Plaintiffs bring this action to seek damages for, and to ultimately stop, Defendant's continued infringement of Plaintiffs' patents, including in particular the '250 Patent, '389 Patent,

’045 Patent, ’224 Patent, ’591 Patent, ’844 Patent, ’721 Patent, ’869 Patent, ’244 Patent, ’243 Patent, ’505 Patent, ’932 Patent, and ’123 Patent (collectively, the “Asserted Patents”) (Exhibits 1-13.) As a result of Sophos’s unlawful competition in this District and elsewhere in the United States, Plaintiffs have lost sales and profits and suffered irreparable harm, including lost market share and goodwill.

NATURE OF THE CASE

17. Plaintiffs bring claims under the patent laws of the United States, 35 U.S.C. § 1, *et seq.*, for infringement of the Asserted Patents. Defendant has infringed and continues to infringe each of the Asserted Patents under at least 35 U.S.C. §§271(a), 271(b) and 271(c).

THE PARTIES

18. Plaintiff Webroot, Inc. is the owner by assignment of each of the Asserted Patents.

19. Webroot has launched multiple cybersecurity products incorporating its patented technology, including for example Webroot SecureAnywhere and Evasion Shield.

20. Webroot is a registered business in Texas with multiple customers in this District. Webroot also partners with several entities in this District to resell, distribute, install, and consult on Webroot’s products.

21. Plaintiff OpenText holds an exclusive license to one or more of the Asserted Patents. OpenText is registered to do business in the State of Texas.

22. OpenText is a Delaware corporation and maintains three business offices in the state of Texas, two of which are located in this District, including one in Austin and another in San Antonio. Over 60 employees work in this District, including employees in engineering, customer support, legal and compliance teams, IT, and corporate development. OpenText also has a data center located in this District. OpenText is in the computer systems design and services industry.

OpenText sells and services software in the United States.

23. Defendant Sophos Ltd. is a foreign corporation with its global headquarters at The Pentagon, Abingdon, OX14 3YP, United Kingdom.

JURISDICTION & VENUE

24. This action arises under the Patent Laws of the United States, 35 U.S.C. § 1, *et seq*. The Court has subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

25. This Court has personal jurisdiction over Defendant because Defendant regularly conducts business in the State of Texas and in this district, including operating systems, using software, providing services and/or engaging in activities in Texas and in this district that infringe one or more claims of the Asserted Patents.

26. Defendant Sophos has further, either directly or through its extensive network of reseller and OEM partnerships, purposefully and voluntarily placed its infringing products and/or services into the stream of commerce with the intention and expectation that they will be purchased and used by customers in this District, as detailed below.

27. Venue is proper in this District pursuant to 28 U.S.C. §§ 1391(b) and (c) and 28 U.S.C. § 1400(b) because, upon information and belief, Defendant Sophos is a foreign entity. Sophos has also committed acts of infringement within this District.

28. On information and belief, Sophos is a foreign corporation with significant contacts with this District. As an example, Sophos has entered into license agreements with end-users in Texas covering the Accused Products and their operation in this District. The Sophos Security Suite End User License Agreements all reference Sophos Limited as the rights-holder under the contract. (*See*, *e.g.*, WBR_SOP000176, https://www.sophos.com/en-us/legal/sophos-end-user-license-agreement.aspx.) Thus, Sophos has entered into license agreements with end-users

covering the Accused Products and their operation in Texas and in this District.

29. On information and belief, Sophos relies on a network of partnerships with "resellers, managed service providers and cybersecurity experts" to sell Accused Products, including Intercept X, to its customers in this District, and to instruct and teach customers how to use the Accused Products. (*See*, *e.g.*, WBR_SOP000506, https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx ("Sophos products and services are sold via trusted partners who recommend and implement the right solutions to meet your unique needs.").)

30. On information and belief, Sophos sells, offers for sale, advertises, makes, installs, and/or otherwise provides endpoint security software and security services, including the Accused Products, the use of which infringes the Asserted Patents in this District. Sophos performs these acts directly and/or through its partnerships with resellers and managed service providers in this District. Those partners include, but are not limited to, "Gold" and "Silver" partners consisting of resellers and managed service providers in this District. (*See* WBR_SOP000525 https://partners.sophos.com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1.)

31. On information and belief, Sophos generates sales to end users within the United States and within this District through its partnerships with resellers and managed service providers. (*Id.*)

32. Sophos has sold infringing endpoint security software and provided infringing endpoint security services to customers who have regular and established places of business in this District, which, on information and belief, deploy Sophos's endpoint security software to their endpoint devices and encourages others to install Sophos's antivirus software on their own devices. (*See, e.g.*, WBR_SOP000216, https://security.utexas.edu/education-outreach/anti-virus.)

33. As further detailed below, Sophos's use, provision of, offer for sale, sales, installation, maintenance, support, and advertising of endpoint security software within this District infringe the Asserted Patents. Sophos' partners infringe the Asserted Patents by using, installing, offering for sale, selling, providing support for, and/or advertising Sophos's endpoint security software within this District. Sophos' customers infringe the Asserted Patents by using Sophos' endpoint security software within this District.

34. Sophos and its partners encourage and induce its partners and customers to use the Accused Products in an infringing way at least by making Sophos's endpoint security services available on its website, widely advertising those services, providing applications that allow partners and users to access those services, provides instructions for installing, and maintaining those products, and/or provides technical support to users, and engaging in activities that aid and abet infringement of the Asserted Patents by end-users. (*See* WBR_SOP000222, https://www.sophos.com/en-us/products/endpoint-antivirus.aspx.).

35. Sophos's partners also infringe (directly or indirectly) the Asserted Patents by installing, maintaining, operating, providing instructions and technical support, and/or advertising the Sophos Security Suite including the Accused Products within this District. End-users and Sophos's partner customers infringe the Asserted Patents at least by installing and using Sophos Security Suite software, which performs the claimed methods in the Asserted Patents within this District.

36. Sophos also contributes to infringement of the Asserted Patents by customers and end users of the Accused Products by offering within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, one or more of the methods claimed in the Asserted Patents, constituting a material part

of the inventions claimed, and not a staple article or commodity of commerce suitable for substantial non-infringing uses. Indeed, the Accused Products and the example functionality described below have no substantial non-infringing uses but are specifically designed to practice the methods claimed in the Asserted Patents.

37. Sophos' infringement adversely impacts Plaintiffs and their employees who live in this district, as well as Plaintiffs' partners and customers who live and work in and around this District. On information and belief, Sophos actively targets and offers Accused Products to customers served by Plaintiffs, including in particular customers/end-users in this District.

PLAINTIFFS' PATENTED INNOVATIONS

38. Plaintiff Webroot and its predecessors were all pioneers and leading innovators in developing and providing modern end point security protection, including "community-based" signatureless threat detection process using AI-driven behavior analysis across the entire network to provide "zero-day" protection against unknown threats.

39. The Asserted Patents discussed below capture technology, features, and processes that reflect these innovations, and improve on traditional anti-Malware and network security systems.

Advanced Malware Detection Patents
U.S. Patent Nos. 8,418,250, 8,726,389, and 8,763,123

40. The '250, '389, and '123 Patents are part of the same patent family and generally disclose and claim systems and processes related to real-time and advanced classification techniques for as-yet unknown malware. These patents are collectively known as the "Advanced Malware Detection" Patents. Plaintiff Webroot owns by assignment the entire right, title, and interest in and to the '250, '389, and '123 Patents. Webroot has granted Plaintiff OpenText an exclusive license to the '250, '389, and '123 Patents.

41. The ’250 Patent is entitled “Methods and Apparatus for Dealing with Malware,” was filed on June 30, 2006, and was duly and legally issued by the United States Patent and Trademark Office (“USPTO”) on April 9, 2013. The ’250 Patent claims priority to Foreign Application No. 0513375.6 (GB), filed on June 30, 2005. A true and correct copy of the ’250 Patent is attached as Exhibit 1.

42. The ’389 Patent is also entitled “Methods and Apparatus for Dealing with Malware,” was filed on July 8, 2012, and was duly and legally issued by the USPTO on May 13, 2014. The ’389 Patent claims priority to the same Foreign Application as the ’250 Patent. A true and correct copy of the ’389 Patent is attached as Exhibit 2.

43. The ’123 Patent is also entitled “Methods and Apparatus for Dealing with Malware,” was filed on July 8, 2012, and was duly and legally issued by the USPTO on June 24, 2014. The ’123 Patent is a division of the application which issued as the ’250 Patent and claims priority to the same Foreign Application as the ’250 Patent. A true and correct copy of the ’123 Patent is attached as Exhibit 13.

44. Malware detection systems in use at the time the Advanced Malware Detection Patents were filed identified malware by maintaining a database of signatures identifying known bad objects (*i.e.*, malware). The signature for an object was conventionally made by creating a hash or checksum corresponding to the object file, which uniquely identifies that object. The signature of each object was then compared to the database to look up whether it matches known malware.

45. If the signature of the object is not found in the database, it is assumed safe or alternatively, the whole file is sent for further investigation by a human analyst. The process of further investigation was typically carried out manually or “semimanually” by subjecting the file

to detailed analysis, for example by emulation or interpretation, which can take days given the human involvement that is typically required. (*See*, *e.g*., Exhibit 2, ’389 Patent, 2:9-17.)

46. This approach had significant drawbacks, including that it required considerable effort by the providers of such systems to identify and analyze new malware and generate signatures of objects that are found to be bad after human analysis. Large vendors of anti-malware packages typically employed thousands of human analysts to identify and analyze objects and keep the database of signatures of bad objects reasonably up to date.

47. However, as the volume of network traffic increases, the task of keeping up with identifying suspect objects and investigating whether or not they are bad becomes practically impossible. (*Id*.) It can take days to subject a suspicious file to detailed analysis given the human involvement, and a considerable period of time elapses before a new file is classified as safe or as malware. Thus, the human analysis introduces a time delay where users are exposed and unprotected from the risks posed by previously unidentified malware. (*See* Exhibit 2, ’389 Patent, 2:9-23, 2:63-67.)

48. By contrast, the methods and systems disclosed and claimed in the ’250, ’389, and ’123 Patents perform automatic, sophisticated review (*e.g*., “pattern analysis”) of the actual attributes of a software object or process and the behavior engaged in by, or associated with, that object or process on computers connected to a network.

49. This review enables a determination of “the nature of the object,” (*e.g*., whether it is malicious or not based on review of the object, its behaviors or the activities associated with the object), without requiring a detailed manual analysis of the code of the object itself or relying exclusively on whether it has a signature that matches an extensive database of known malicious “signatures.” (*See* Exhibit 2, ’389 Patent, 3:14-24; Exhibit 1, ’250 Patent, 3:7-18.) This provides a

significant improvement to the operation of the computer network because monitoring behavior or other information about the object or process, rather than code or signature matching, allows the system to rapidly determine the nature of the object (*e.g.*, malware), without requiring a detailed manual analysis of the code of the object itself as in conventional anti-virus software. (*See* Exhibit 1, '250 Patent, 3:11-18.)

50. The approaches in the Advanced Malware Detection Patents are generally focused on receiving information about the behavior of objects or processes on remote computers at a base computer. This information is analyzed automatically by, for example, mapping the behavior and attributes of objects known across the community in order to identify suspicious behavior and to identify malware at an early stage. This approach allows, among other advantages, the number of human analysts needed to be massively reduced. It also improves the computer network by reducing the latency involved with identifying new threats and responding to objects exhibiting new, potentially malevolent behavior. (WBRT001252 at WBRT001663-1664, '250 Patent Prosecution History, 2010-09-07 Amendment at 16-17.)

51. Each of the claimed inventions of the Advanced Malware Detection Patents is necessarily rooted in computer technology—in other words, the identification of malicious computer code in computer networks is fundamentally and inextricably a problem experienced with computer technology and networks—and addresses this fundamental computer technology problem with a computer technology solution. Furthermore, the Advanced Malware Detection Patents improve the technical functioning of the computer network using techniques—such as analyzing behavioral information about or associated with computer objects and processes—to improve network security by identifying malware more quickly and with less resources. These technical improvements address identified weaknesses in conventional systems and processes.

(*See*, *e.g.*, Exhibit 1, '250 Patent, 2:5-3:18.)

52. In particular, the '250 Patent describes and claims methods and systems that include receiving behavioral data about or associated with a computer object from remote computers on which the object or similar objects are stored; comparing in a base computer the data about the computer object received from the remote computers; and, classifying the computer object as malware on the basis of said comparison if the data indicates the computer object is malware. In effect, this process builds a central picture of objects and their interrelationships and activities across the entire community and allows automation of the process of identifying malware by aggregating and comparing the activity of objects running across the community (*i.e.*, on multiple remote computers).

53. The '250 Patent further provides that a mask is automatically generated for an object that defines "acceptable behavior" for the object. The operation of the computer object is then monitored and if the actual monitored behavior extends beyond that permitted by the mask, the object is disallowed from running and reclassified as malware.

54. The claimed methods and systems of the '250 Patent constitute technical improvements over the traditional anti-malware systems and provide numerous advantages to computer systems and the process of detecting malware. In addition to the advantages set forth above, the methods and systems claimed in the '250 Patent provide additional advantages in dealing with objects that do not initially exhibit suspicious behavior, but later start to exhibit malevolent behavior. Traditional malware systems could only mark a computer object as good or bad (*i.e.*, a binary decision), and did so by examining the signature of the object itself against a database of "known bad" signatures. This approach does not permit the system to automatically deal with the case where an object does not initially exhibit suspicious behavior but starts to exhibit

malevolent behavior in the future.

55. By contrast, the ’250 Patent improves these systems by generating an appropriate behavior mask for the object and then continuing to monitor the behavior of the object. If the object operates out of bounds of the permitted behavior, then an appropriate action is taken, such as disallowing the computer object from running and reclassifying the object as malware. Thus, the systems and methods described and claimed further the operation and security of the network by stopping an object from running and changing the classification of an object in real-time when unacceptable behavior is identified. (*See* Exhibit 1, ’250 Patent, 3:47-50; 4:19-30.)

56. Furthermore, the methods and systems claimed in the ’250 Patent, including generating a “mask” of acceptable behavior, allowing an object to run, continuing to monitor the object, and disallowing/reclassifying the object if the behavior extends beyond that permitted by the mask, are not routine or conventional. For example, while a “safe,” mask-permitted version of notepad.exe “would not be expected to perform a wide variety of events, such as transmitting data to another computer or running other programs or running other programs” a “modified” and potentially “malevolent” version of notepad.exe could perform those unexpected events. (*See id.* at 11:27-41.) Unlike traditional malware systems that would have already made a binary determination that the notepad.exe object is safe, the methods and systems of the ’250 Patent re-classify that version of notepad.exe as malware when its behavior becomes unexpected and “extends beyond that permitted by the mask.” (*Id*. at 4:19-30.)

57. The applicants provided another example illustrating the unconventional nature and technical advantages and improvements, offered by the claimed systems and methods during prosecution:

58. As an example, suppose a new version of Internet Explorer appeared. This could

be a legitimate update to Internet Explorer released by Microsoft or alternatively it could be a file infected with a virus. In the prior art, the new object would have an unknown signature, so an in-house analyst would laboriously analyze the new object and determine whether or not it was safe. Whilst this analysis is carried out, the object would either be blocked, which would cause huge inconvenience to users of the new object, or allowed to run, in which case there is a risk of the object performing malevolent acts. In contrast, the present invention would collect data at the base computer from remote computers running the new version of Internet Explorer. Using the information collected, the system could determine that the new object purports to be a new version of Internet Explorer. However, it may not be apparent at this point whether or not the new object is capable of malevolent behaviour. In this scenario the present invention generates an appropriate behavioural mask for the object, *e.g*. by using a profile of behaviour of previous versions of Internet Explorer that are known not to be malware, or by using a profile for the behaviour appropriate for a web browser. The remote computers are allowed to let the new version run whilst monitoring its behaviour against the mask. The instant the new object exhibits some new, malevolent behaviour, this can be stopped at the remote computer, as well as being flagged to the base computer and used at the base computer to change the classification of the object. Thus, the present invention allows an instant response to an object changing its behaviour to exhibit malevolent behaviour in the future. (*See* WBRT001252 at WBRT001665-1666, ’250 Patent Prosecution History, 2010-09-07 Amendment at 18, 19.)

59. Similarly, the ’389 Patent describes and claims deploying an unconventional “event” based model that classifies a particular object as malicious or safe by analyzing real-time data sent by remote computers on the events, or actions, that a particular software “object,” and other objects deemed similar to it, initiate or perform on those computers. (*See* Exhibit 2, ’389

Patent, 3:14-55.) This information is collected from across the network, correlated and used for subsequent comparisons to new or unknown computer objects to identify relationships between the correlated data and the new or unknown computer objects. The objects may be classified as malware based on this comparison.

60. Through continuous aggregate analysis of events involving computer objects as they occur across network endpoints, the methods and systems described and claimed in the ’389 Patent maintain up-to-date information about computer objects (including malicious objects) seen across the network, identify relationships between those previously identified objects and any new or unknown objects, and make malware determinations based on those relationships. “For example, a new object that purports to be a version of notepad.exe can have its behavior compared with the behav[io]r of one or more other objects that are also known as notepad.exe … In this way, new patterns of behav[io]r can be identified for the new object.” (*Id.* at 10:58-65.)

61. The methods and systems described and claimed in the ’389 Patent can rapidly determine “the nature of the object,” (*e.g.*, whether it is malicious or not) based on information such as the behavior of the object or effects the object has, without requiring “detailed analysis of the object itself as such” (manually reviewing the object’s code) or reliance on matching an extensive database of known malicious “signatures.” (*Id.* at 3:14-24; Exhibit 1, ’250 Patent, 3:7-18.)

62. The ’123 Patent generally is directed to the real-time and cloud-based determination of whether a particular computer is vulnerable to a particular malware process. The ’123 Patent bases this determination on the security products it has installed and the products’ vulnerabilities.

63. The methods of the ’123 Patent therefore are directed to solutions to specific problems that arose in, and are necessarily rooted in, computer technology. Prior art models

performed simple comparisons between the versions of software installed (*e.g.*, an internet browser) on a given computer and their "known vulnerabilities" in a static database. (Exhibit 13, '123 Patent, 2:52-57.) These models were limited in that they only performed simple comparisons of what software applications a particular computer had installed (*e.g.*, a particular internet browser), to the vulnerabilities with which that application was installed in a static database.

64. By contrast, the methods described in the '123 Patent target vulnerabilities in an end user's computer caused by blind spots in its locally installed security products. The described methods then identify those vulnerabilities in real time by marshalling a continuously updating "community database." The "community database" holds historical information, continually updating vulnerability data from "many, possibly millions, of users' computers," including the "configuration of security products" and operating systems each of those millions of computers had installed, and which such combinations were "susceptible or vulnerable to any particular malware object." (*Id.* at 16:22-50, 17:5-15.)

65. To reduce "the data quantity for storage and transmission," the community database receives this configuration information in the form of a highly compressed "signature or key" that encapsulates all "the details of all local security products, versions, signature files, firewall settings, etc." (*Id.* at 16:24-31.)

66. Storing "keys" in such a form enables quick searches of the community database to find the configuration "key" of any particular computer and associate that key with the vulnerabilities to which that computer is exposed. (Exhibit 13, '123 Patent, 16:22-39.) The search, diagnosis and resulting message to the user (*e.g.*, "directing the user to a website" from which she can download a particular security update) are performed "virtually in real-time." (*Id.* at 17:5-15.)

67. The Advanced Malware Detection Patents provide systems and methods that

necessarily address issues unique to computer networks and computer network operation; namely the identification of "bad" software (*e.g.*, malware, viruses, etc.). These patents all provide unique network security enhancement that solves the technical problem of rapidly identifying newly arising and emerging malware by reviewing information about the object and processes (*e.g.*, the behaviors and events associated with software objects and processes running on computers within the network).

68. The systems and methods claimed in the Advanced Malware Detection Patents improve the operation of computer networks by identifying malicious objects in real-time and taking action to remove or eliminate the threat posed by the malware object or process once it has been identified. The claimed inventions in these patents provide a technological solution to a technological problem—the inability of conventional code or signature matching solutions to identify new or unknown malware objects or processes at or near the runtime of the objects or processes themselves without the extensive delay and resource use associated with traditional systems

Forensic Visibility Patents
U.S. Patent No. 9,578,045 and U.S. Patent No. 10,257,224

69. The '045 and '224 Patents are part of the same patent family and are each generally directed to providing forensic visibility into computing devices in a communication network by analyzing network events and creating audit trails. Plaintiff Webroot owns by assignment the entire right, title, and interest in and to the '045 and '224 Patents. Webroot has granted OpenText an exclusive license to the '045 and '224 Patents.

70. The '045 Patent is entitled "Method and Apparatus for Providing Forensic Visibility into Systems and Networks," was filed on May 5, 2014, and was duly and legally issued by the USPTO on February 21, 2017. The '045 Patent claims priority to provisional application

61/819,470 filed on May 3, 2013. A true and correct copy of the ’045 Patent is attached as Exhibit 3.

71. The ’224 Patent is also entitled “Method and Apparatus for Providing Forensic Visibility into Systems and Networks,” was filed on February 20, 2017 and was duly and legally issued by the USPTO on April 9, 2019. The ’224 Patent claims priority to the ’045 Patent and also to provisional application 61/819,470 filed on May 3, 2013. A true and correct copy of the ’224 Patent is attached as Exhibit 4.

72. The ’045 and ’224 Patents describe and claim inventive and patentable subject matter that significantly improves on traditional network forensic tools used to discover or identify security issues on computer networks. Network forensics generally relates to intercepting and analyzing network events to discover the source of security attacks. (*See* Exhibit 3, ’045 Patent, 1:22-24; Exhibit 4, ’224 Patent, 1:24-26.)

73. The ’045 and ’224 Patents improved on prior art network forensics tools by providing a technical solution to a technical problem experienced by computer networks and computer network operation. Unlike traditional network forensic tools, these patents create forensic visibility into the computing devices on the communication network to identify malware or other security issues in operation of those devices. (*See* Exhibit 3, ’045 Patent, 2:36-38; Exhibit 4, ’224 Patent, 2:38-40.)

74. In particular, the Forensic Visibility Patents improve network security by gathering an “event,” generating “contextual state information,” obtaining a “global perspective” for the event in comparison to other events, and generating/transmitting an “event line” that includes information for the event. (*See* Exhibit 3, ’045 Patent, cl. 1; Exhibit 4, ’224 Patent, cl. 1.) The described and claimed systems and methods intercept network events, create audit trails, or

contextual states, for each individual event by correlating the event to objects such as their originating processes, devices, and/or users, and establishing a global perspective of the objects. The claimed systems and methods of the Forensic Visibility Patents address an identified weakness in conventional systems and processes; namely the ability to monitor, capture and/or analyze what is occurring *at* computing devices on a computer network, thereby providing an improved way to address the technical problem of discovering security attacks or security problems within a computer network.

75. In addition to analyzing the behavior of an object to identify those that are potentially malicious, malware detection is further improved by understanding the context of the event and computer objects of interest. (*See* Exhibit 3, ’045 Patent, 2:39-45 (“The system filters may be built upon the same or similar technology related to behavior monitoring and collection, as discussed in U.S. application Ser. No. 13/372,375 filed Feb. 13, 2012, (Methods and Apparatus for Dealing with Malware”)).) In particular, in many cases a potentially malicious object is identified by the system as a result of other events that provide information as to whether the code is malicious. For example, if an object or event under investigation originated from an object or event that is known to be malicious or have malicious behaviors or characteristics, the presence of the known, malicious object provides a further indication that the potentially malicious object or event is malicious as well.

76. The patents further explain that in addition to context information, the systems and techniques can also use information from the network to obtain a global perspective of the network operation. The combination of contextual information and global perspective enables detection of new zero-day threats, including objects created from objects (or similar objects) that have been identified previously as malicious. Indeed, in the context of modern computers and network

systems that generate tens of millions of events every minute, the use of a global perspective and contextual information to correlate an event or object under investigation with prior, related events and objects—including the originating object—significantly improves the ability of the system to identify potential threats.

77. The patents further disclose technical improvements to forensic systems by "assembling" or "generating" an "event line" based on the contextual information—including the correlation to the originating object—and global perspective. (*See, e.g.*, Exhibit 3, '045 Patent, 9:50-58.) The generation of the event line makes it easier for end users to "identify events, and/or instances of malware, that require more immediate attention"—thereby improving the accuracy and efficiency of identifying additional malicious code, as well as enabling administrators to more readily analyze malware, assess vulnerabilities, and correct damage done by the originating objects (and other objects in the event chain). (*See* Exhibit 3, '045 Patent, 9:45-49.) The generation and use of an event line itself was, at the time, an unconventional way in which event information, contextual state information and global perspectives are generated, communicated, and/or potentially displayed to, and interacted with by, an administrator or end user.

78. Thus, the '224 and '045 Patents describe and claim systems and methods that provide technical advantages and improvements over traditional network security and forensic systems, including more efficient and accurate identification of malware (*e.g.*, the contextual and global perspective information reduced false negative and positives for malware detection). The described systems and methods also improved the identification of other malware (and corresponding events) that might otherwise go undetected in prior systems, thereby improving system performance and reducing the number of resources required.

79. Indeed, the described systems and methods enable end-to-end forensic visibility into event occurrences across a networked environment and from the bottom of the stack to the top, thereby improving upon conventional network forensic products. (*See* Exhibit 3, ’045 Patent, 2:31-38, 3:49-55; Exhibit 4, ’224 Patent, 2:33-40, 3:52-59; *see also* Exhibit 3, ’045 Patent, 4:36-41; Exhibit 4, ’224 Patent, 4:39-44.)

80. Applicant further explained during prosecution how the generation of contextual state information and obtaining a global perspective—including for objects and events other than those that were detected, such as the originating object—are unconventional steps in the areas of malware detection and network forensics. For example, Applicant explained how the described systems and methods improve the system performance of computing devices:

> In this case, the claimed invention provides for determining correlations between events and objects and creating an audit trail for each individual event. For example, a context analyzer may correlate an actor, victim, and/or event type to one or more originating processes, devices, and users. After the analysis is complete, a sensor agent may use the correlated data to generate a global perspective for each event such that an administrator is able to forensically track back any event which occurs to what triggered it. Thus, the global perspective represents a drastic transformation of raw event data into a comprehensive, system-wide forensic audit trail. (*See* WBRT005145, ’045 Patent Prosecution History, 2016-03-16 Amendment at 11-12.)
>
> In this case, examples of the claimed systems and methods provide low level system filters which intercept system events “in a manner such that the operation of the system filter does not impact system performance.” Specification, para. [0008]. For example, on an average system, because tens of millions of events take place every minute, the noise ratio can prevent forensic solutions from being able to provide sufficient value to the end consumer of their data due to the inability to quickly find important events. A product which impacts system performance will have considerably diminished value to an administrator and can negatively affect the results of an analysis undertaken. Examples of the present systems and methods address this shortcoming by providing a system filter that substantially improves the system performance of the computing devices in the system. (*See* WBRT005145, ’045 Patent Prosecution History, 2016-03-16 Amendment at 12.)

81. During prosecution, Applicant further explained how the claims are directed to solving a technical problem and a specific improvement in computer functionality relating to computer security:

> ***[T]he claims are directed to solving a technical problem***. Typically, network forensic systems use network forensic tools (*e.g.*, network sniffers and packet capture tools) to detect and capture information associated with communication sessions. Although such network forensic tools are operable to passively collect network traffic, the tools reside at a network edge (*e.g.*, outside of a system or hosts). As a result, the network forensic tools have no ability to obtain useful information within a host or to establish any sort of context from within a host that is generating and/or receiving network events. To address this, aspects of the present disclosure enable methods for providing forensic visibility into systems and networks. For example, a local aggregator/interpreter, context analyzer and sensor agent may provide visibility into occurrences across an environment to ensure that a user (*e.g.*, an administrator) is aware of any system change and data communications in and out of the computing devices residing on the network. During this process, identified events may be correlated to objects, thus creating an audit trial [sic] for each individual event. (*See* WBRT005145, '045 Patent Prosecution History, 2016-03-16 Amendment at 9-10. (emphasis added))
>
> Here, ***the claims are directed to a specific improvement in computer functionality relating to computer security, and more specifically to providing end-to-end visibility of events within a system and/or network***. (*See* WBRT006334, '224 Patent Prosecution History, 2018-08-29 Amendment at 10-11 (citing '224 Patent specification) (emphasis added).)
>
> The Specification subsequently discusses a variety of ways in which the claimed subject matter solves the above-described problem. For example: "It is, therefore, one aspect of the present disclosure to provide a system and method whereby events occurring within a computing device are captured and additional context and a global perspective is provided for each capture event. For example, a sensor agent may provide visibility into occurrences across an environment, such as a networked environment, to ensure that an administrator is aware of any system changes and data communication in and out of computing devices residing on the network." (*See* WBRT006334, '224 Patent Prosecution History, 2018-08-29 Amendment at 11-12 (citing '224 Patent specification).)

82. In response to these arguments, the Examiner withdrew a rejection based on 35 U.S.C. §101 and allowed the claims of the Forensic Visibility Patents to issue. As recognized by

the USPTO Examiner, the claimed inventions of the ’045 and ’224 Patents provide a technical solution to the technical problem of forensic visibility regarding events in a computer network.

US. Patent No. 10,284,591

83. U.S. Patent No. 10,284,591 is entitled “Detecting and Preventing Execution of Software Exploits,” was filed on January 27, 2015 and was duly and legally issued by the USPTO on May 7, 2019. The ’591 Patent claims priority to provisional application 61/931,772 filed January 27, 2014. A true and correct copy of the ’591 Patent is attached as Exhibit 5. Plaintiff Webroot owns by assignment the entire right, title, and interest in and to the ’591 Patent. Webroot has granted Plaintiff OpenText an exclusive license to the ’591 Patent.

84. The ’591 Patent describes and claims an “anti-exploit” technique to prevent undesirable software and/or other computer exploits from executing. (*See* Exhibit 5, ’591 Patent, 1:13-28, 1:32-33.) Computer “exploits” include code, software, data, or commands that take advantage of a bug, glitch, or vulnerability in a computer system. To accomplish this goal, the novel anti-exploit techniques described and claimed in the ’591 Patent monitors memory space of a process for execution of functions and performs “stack walk processing” upon invocation of a function in the monitored memory space. (*Id.* at 1:33-39.) During that stack walk processing, a memory check may be performed to detect suspicious behavior. (*Id.*) If the memory check detects certain types of suspicious behavior, an alert may be triggered that prevents the execution of a payload for the invoked function. (*Id.* at 1:39-48.)

85. The ’591 Patent describes and claims unconventional “stack walk processing” techniques for detecting and preventing unwanted software exploits during which memory checks are performed before an address of an originating caller function is reached. The anti-exploit techniques can include performing “memory checks performed during the stack walk processing once an address is reached for an originating caller function.” (*Id.* at 8:6-7.) In one embodiment,

"memory checks from the lowest level user function of the hooked function down through the address of the originating caller function" may be performed to detect and identify suspicious behavior. (*Id.* at 6:7-11.)

86. The "stack walking" and "memory checks" described and claimed in the '591 Patent are fundamentally rooted in computer technology—in fact, they are processes only performed within a computer context. The techniques described and claimed in the '591 Patent addresses a problem that specifically arises in the realm of computer technology (namely, computer exploit identification) by, *inter alia*, performing memory checks and detection specified behavior during stack walking.

87. The '591 Patent further describes and claims unconventional techniques that address identified weaknesses in conventional exploit prevention technologies. For example, unlike exploit prevention technologies that try to prevent an exploit from ever starting its own shellcode to execute a malicious payload, the '591 Patent describes and claims techniques that prevent shellcode from executing a malicious payload even if the shellcode has been started. (*Id.* at 6:24-30; *see also* 7:56-62.) Thus, these unconventional techniques address an identified weakness in conventional exploit prevention systems and provide technical advantages including enhanced security protection, improved detection of potential security exploits, reduction in error rate identifying and marking suspicious behavior (*e.g.*, false positives), and improved usability and interaction for users who are not required to continuously monitor for security exploits. (*Id.* at 2:44-51.) As such, the '591 Patent describes and claims specific computer-related technological steps to accomplish an improvement in computer security and functionality and is directed to a specific technological solution to a problem unique to computers.

U.S. Patent Nos. 10,599,844 and 11,409,869

88. The ’844 Patent is entitled “Automatic Threat Detection of Executable Files Based on Static Data Analysis,” was filed May 12, 2015 and was duly and legally issued by the USPTO on March 24, 2020. A true and correct copy of the ’844 Patent is attached as Exhibit 6. Plaintiff Webroot owns by assignment the entire right, title, and interest in and to the ’844 Patent. Webroot has granted Plaintiff OpenText an exclusive license to the ’844 Patent.

89. The ’844 Patent is entitled “Automatic Threat Detection of Executable Files Based on Static Data Analysis,” was filed May 12, 2015 and was duly and legally issued by the USPTO on March 24, 2020. A true and correct copy of the ’844 Patent is attached as Exhibit 6. Plaintiff Webroot owns by assignment the entire right, title, and interest in and to the ’844 Patent. Webroot has granted Plaintiff OpenText an exclusive license to the ’844 Patent.

90. The ’869 Patent, entitled “Automatic Threat Detection of Executable Files Based on Static Data Analysis,” is a continuation of the ’844 Patent and claims priority to the application of the ’844 Patent. The ’869 Patent was filed on February 14, 2020 and was duly and legally issued by the USPTO on August 9, 2022. A true and correct copy of the ’869 Patent is attached as Exhibit 8. Plaintiff Webroot owns by assignment the entire right, title, and interest in and to the ’869 Patent. Webroot has granted Plaintiff OpenText an exclusive license to the ’869 Patent.

91. The ’844 and ’869 Patents address and improve upon conventional approaches to malware detection in computer networks and computer network operation. Every day, an uncountable number of new executable files are created and distributed across computer networks. Some of those files are unknown, and malicious. It is, thus, vital to accurately and immediately diagnose those files for any potential threat, while also efficiently using resources (*e.g.*, processing power). (*See* Exhibit 6, ’844 Patent, 1:7-13.)

92. Conventional approaches for diagnosing potential malware threats were costly and time consuming, making it difficult to realistically address zero-day threats for all of the files entering a system. These "[a]pproaches to detecting threats typically focus[ed] on finding malicious code blocks within a file and analyzing the behavior of the file." (*See* Exhibit 6, '844 Patent, 2:15-17.) Encrypted files would be decrypted then disassembled to extract the code for analysis, typically by traditional anti-virus software based on signature matching. (*Id.* at 2:15-20) If the code was malware, investigating its behavior involved running the code on the system, which put the system at risk. (*Id.* at 2:20-23.)

93. Another approach for protecting against potential threats from unknown executable files involved wavelet decomposition to determine software entropy. (*See* WBRT007115 at 7491, '844 Patent Prosecution History, April 24, 2019 Applicant Remarks, at 8.) Wavelet decomposition is a process where an original image is decomposed into a sequence of new images, usually called wavelet planes. (*Id.*) In this method, each data file in a set of data files is split into random, non-overlapping file chunks of a fixed length. (*Id.*) Those file chunks are then represented as an entropy time-series, which measures the time it takes for each chunk to decompose. (*Id.*) Said differently, this approach measured how much time it took a data file to decompose. (*Id.*) Once the file decomposition rate, or entropy time-series, had been calculated, that rate would be compared to decomposition rates of "known bad" files to identify files that contain malware. (*Id.* at WBRT007492.) This process required significant computing resources—typically taking hours to complete—and was not sufficiently accurate in identifying malware.

94. The '844 and '869 Patents significantly improve upon and address shortcomings associated with these prior approaches. The Patents describe and claim methods and systems that detect threats in executable files without the need to decrypt or unpack those executable files by

extracting "static data points inside of the executable file without decrypting or executing the file," generating "feature vectors" from those static data points, selectively turning on or off features of the feature vector, and then evaluating the feature vector to determine if the file is malicious. (*See*, *e.g.*, Exhibit 6, '844 Patent, 1:20-21; cl. 1.) The described systems and methods enable accurate and efficient identification of malware without the need to distinguish between encrypted files and non-encrypted files (*id.* at 6:58-59), thereby significantly increasing efficiency and reducing processing resources required to analyze each potentially malicious computer object. By using this unconventional approach to determine whether a file executable on a computer poses a threat, the '844 and '869 Patents improve on the operation of the computer network associated with the computer by enhancing security, including by increasing detection of new threats, reducing the error rates in identifying suspicious files, and improving efficiency in detecting malicious files. (*See* Exhibit 6, '844 Patent, 2:46-56.)

95. The '844 and '869 Patents describe and claim techniques that employ a learning classifier (*e.g.*, a machine-learning classifier) to determine whether an executable file is malicious, for example by using the classifier to classify data into subgroups and identify and analyze specific data points to which those subgroups correspond. (*See* Exhibit 6, '844 Patent, 4:33-41, 7:40-8:1.) The described and claimed techniques also selectively turn on or off features for evaluation by the learning classifier. (*See id*. at 7:57-66.) Doing so accelerates analysis and reduces false positives by testing those features of a file likely to be relevant to a determination of its maliciousness. For example, the learning classifier "may detect that the file does not contain 'legal information'," such as "timestamp data, licensing information, copyright information, etc." (*See id.* at 7:66-8:5.) In this example, given the lack of legal protection information in the file, the learning classifier would "adaptively check" the file for additional features that might be indicative of a threat," while

"turn[ing] off," and thus not use processing time unnecessarily checking features related to an evaluation of "legal information." (*Id.* at 8:5-10.)

96. Second, the '844 and '869 Patents describe and claim techniques that use character strings extracted from within the executable file to generate a feature vector and then evaluate that feature vector using support vector processing to classify executable files. (*See* Exhibit 6, '844 Patent, 9:2-11.) The classifier provides, for example, the ability to leverage the indicia of "benign" files, which use "meaningful words" in certain data fields, versus "malicious" files, which leave such fields empty or full of "random characters," to build meaningful feature vectors that are analyzed to make faster and more identifications of malware (*See*, *e.g.*, Exhibit 6, '844 Patent, 9:2-18.)

97. The '844 and '869 Patents are thus directed to specific solutions to problems necessarily rooted in computer technology, namely, the determination whether a file executable on a computer poses a threat. The '844 and '869 Patents improved upon the accuracy and efficiency of malware detection. (*See* Exhibit 6, '844 Patent, 2:15-45.)

98. By using some or all of the unconventional techniques described above to determine whether a file executable on a computer poses a threat, the '844 and '869 Patents address a problem necessarily involving computers and improve upon the operation of computer networks. In particular, the '844 and '869 Patents achieve a number of technical advantages over conventional approaches to malware detection including, for example:

- enhanced security protection including automatic detection of threats, reduction or minimization of error rates in identification and marking of suspicious behavior or files (*e.g.*, cut down on the number of false positives),

- ability to adapt over time to continuously and quickly detect new threats or potentially unwanted files/applications,
- improved efficiency in detection of malicious files, and
- improved usability and interaction for users by eliminating the need to continuously check for security threats.

(*See* Exhibit 6, ’844 Patent, 2:15-57.)

U.S. Patent No. 9,413,721

99. The ’721 Patent is entitled “Methods and Apparatus for Dealing with Malware,” was filed on February 13, 2012, and duly and legally issued by the USPTO on February 5, 2013. A true and correct copy of the ’721 Patent is attached as Exhibit 7. Plaintiff Webroot owns by assignment the entire right, title, and interest in and to the ’721 Patent. Webroot has granted Plaintiff OpenText an exclusive license to the ’721 Patent.

100. The systems and methods described and claimed in the ’721 Patent are directed to improved techniques for detecting and classifying malware, a technological problem fundamentally and inextricably associated with computer technology and computer networks. The ’721 Patent explains that prior anti-malware products used signature matching to detect malware, either locally or at a central server. (Exhibit 7, ’721 Patent, 1:37-2:14.) The local anti-malware product suffered from delays in identifying new malware threats and obtaining signatures for them so they could be blocked. (*Id.* at 1:37-55.) Central servers stored signatures in the cloud. (*Id.* at 56-57.) But only signature or very basic information was sent to the central server for matching. (*Id.* at 1:67-2:2.) If the object was unknown, a copy had to be sent to the central server for investigation by a human, a time consuming and laborious task. (*Id.* at 2:5-7.) In a network environment, it was unrealistic for a human to investigate each new object due to the high volume of incursions that

take place over a network. (*Id.* at 2:7-10.) Thus, under these approaches, “malevolent objects may escape investigation and detection for considerable periods of time.” (*Id.* at 2:10-13.)

101. To address these shortcomings, the ’721 Patent describes and claims unconventional, novel distributed system architectures, such as remote computers that may be allocated to “threat” servers, with “central” servers sitting behind them. (Exhibit 7, ’721 Patent, 9:16-57.) These enhanced computer architectures provide a technical solution to the technical problem of detecting and classifying malware in a computer network environment, thus improving network security while identifying and classifying malware threats in real-time without delays engendered by use of human analysts. (*See, e.g.*, Exhibit 7, ’721 Patent, 1:60-2:7)

102. In particular, the ’721 Patent described and claims embodiments that may include three-tiered architectures of remote computers, threat servers, and a central server that provides a technical enhancement to the computer network itself (improving upon the two-tiered architectures of traditional systems having only remote computers and a central server) by enabling the central server to keep a master list “of all data objects, their metadata and behaviour seen on all of the remote computers” and propagate it back to the threat servers. (*Id.* at 12:28-54.) This novel network architecture improves the operation of the computer network over traditional networks because, for example and as described in the ’721 Patent, “[t]his scheme has been found to reduce workload and traffic in the network by a factor of about 50 compared with a conventional scheme.” (*Id.* at 12:55-57.)

103. Further, “by being able to query and analyze the collective view of an object, *i.e.*, its metadata and behaviours, across all agents [] that have seen it, a more informed view can be derived, whether by human or computer, of the object. In addition, it is possible to cross-group objects based on any of their criteria, *i.e.* metadata and behaviour.” (*Id.* at 18:17-22.) Thus,

embodiments enable better malware identification than conventional systems (*e.g.*, using human analysis) in addition to providing an efficiency benefit. The patent explicitly notes that "the work in processing the raw data [] is too large of a task to be practical for a human operator to complete." (*Id.* at 18:50-52.)

104. The systems and methods described and claimed in the '721 Patent provide further technical improvements. For example, the information collected at the central server includes addition information about the object being classified as well as a count associated with the number of times that the first computer object has been seen to the central server. (*Id.* at cl. 1.) As explained above, using information about the object (such as behavior information) being classified, the systems and methods described and claimed in the '721 Patent provide an approach that is more effective than traditional code or signature matching techniques for classifying objects as malicious. (*Id.* at 1:54-2:14.)

105. Prior methods of classifying malware had technical drawbacks when used on a distributed network. For example, a distributed network that required each server to maintain rules for determining what is malware required each server to deal with huge amounts of largely common data. (Exhibit 7, '721 Patent, 12:20-24.) It was also generally impractical to store the required data on each server because, for example, there were problems determining whether or not the data—which is both massive and constantly changing—is common and up-to-date in real-time. (*Id.* at 12:24-27.) The 3-tiered architectures described and claimed in embodiments of the '721 Patent provides a technical solution for distributed computer networks by, *inter alia*, reducing the workload across the network. (*Id.* at 12:28-59.)

106. Accordingly, the '721 Patent discloses and claims, among other things, an unconventional technological solution to the inherently computer-network centric technical issue

of identifying malware in computer systems. The solution implemented by the ’721 Patent provides a specific and substantial improvement over prior malware classification systems, for example by introducing novel computer network architecture elements combined in an unconventional manner. These approaches improve the function and working of malware detection services by, for example, utilizing multiple threat servers and central servers and performing the analysis and communication carried out by each type of server in an unconventional and efficient manner. (*See, e.g.*, Exhibit 7, ’721 Patent, cl. 1.) These elements and their combination represent a marked improvement in the functioning of computer systems utilized to identify and detect malware in computers networks.

U.S. Patent No. 8,856,505

98. U.S. Patent No. 8,856,505 (the "’505 Patent"), entitled "Malware Management Through Kernel Detection During a Boot Sequence," was filed on April 30, 2012 and was duly and legally issued by the USPTO on October 7, 2014. The ’505 Patent claims priority to, and is a continuation of, U.S. Patent No. 8,190,868 filed August 7, 2006. A true and correct copy of the ’505 Patent is attached as Exhibit 11. Plaintiff Webroot owns by assignment the entire right, title, and interest in and to the ’505 Patent. The ’505 Patent describes and claims techniques to prevent rootkits, spyware, and/or other undesirable software from executing. (*See* Exhibit 11, ’505 Patent, 2:17-29, 4:3-5.). In general, periodic scanning of a computing system often fails to identify pestware, much less stop pestware from executing. (*Id*. at 1:61-67.) The intervening time between periodic scans permits pestware to cloak and execute, leaving the infected computing system exposed, and ultimately vulnerable to the collection and reporting of information held thereon to the malicious third-party. (*Id*.) And pestware that loads relatively early in the boot sequence may

be generally undetectable (*e.g.*, cloaked) when scanned later in the boot sequence or otherwise. (*Id*.)

99. The pestware management techniques of the ’505 Patent mitigate these, and other challenges associated with pestware detection and mitigation, including pestware that executes early in the boot sequence and/or that may be undetectable or cloaked once a computing system runs native applications. (*Id*. at 2:17-29, 3:39-46.) The ’505 Patent describes and claims techniques that can monitor and identify pestware events early in the boot sequence of a computing system. (*Id*. at 3:19-31.) As additional drivers and applications are loaded in the course of the boot sequence, techniques that are described and claimed can monitor the loadings and other events and facilitate management of the pestware. (Exhibit 11, ’505 Patent, 4:19-31, 4:53-62). Thus, the pestware management techniques may operate to stop pestware before it cloaks and executes, including stopping pestware which may otherwise be undetectable by conventional, periodic, post-boot-sequence scans.

100. The ’505 Patent describes and claims an unconventional “kernel-level monitor” to facilitate the management of pestware early in the boot sequence of a computing system. (*Id*. at 3:1-14.) The kernel-level monitor “begins early in the boot and operating system loading process,” (*Id*. at 3:25-27), and “is responsible for detecting pestware or pestware activity on a protected computer or system,” (*Id*. at 3:19-21). Attempts by pestware to modify the operating system are monitored and intercepted by the kernel-level monitor, for example, by using “driver hooks and monitoring mechanisms,” which may be placed into the operating system kernel during an early portion of the boot sequence. (*Id*. at 3:32-39, 4:53-62.) Accordingly, the kernel-level monitor can be configured to identify changes at the kernel-level (*e.g.*, such as those changes which may be

attempted by a rootkit or other malicious component), prior to the time when those changes may become generally undetectable. (*Id*. at 4:53-62, 5:1-12.)

101. The ’505 Patent further describes and claims unconventional techniques that permit scanning of the “registry” during the boot sequence “before most services are loaded and executed.” (*Id*. at 6: 4:10.) Performing this type scanning early in the boot process provides new and unexpected benefits, including providing additional information that may be utilized in the identification of malicious programs. For example, scanning the protected parts of the computer such as the registry early in the boot process enables later “examin[ation] and utiliz[ation]” of that information to, for example, “generate new behavior rules for the kernel-level monitor.” (*Id*. at 6:4-10.)

102. The ’505 Patent describes and claims unconventional techniques that address technical weaknesses in conventional mitigation techniques for pestware. (Exhibit 11, ’505 Patent, 1:46-66, 2:1-29.) The ’505 Patent describes and claims techniques that permit pestware management at the kernel level, early in the boot sequence before the malicious code can cause damage or conceal itself. (*Id*. at 2:17-29, 4:63-67, 5:1-12.) Thus, these novel and unconventional techniques provide technical advantages such as enhanced security protection, improved detection of pestware, and adaptability to evolving pestware threats. (*Id*. at 1:46-66, 2:1-29.) As such, the ’505 Patent describes and claims specific computer-related technological steps to accomplish an improvement in computer security and functionality and is directed to a specific technological solution to a problem unique to computers.

Kernel-Level API Monitoring Patents
U.S. Patent Nos. 8,201,243 and 8,719,932

103. The ’243 and ’932, Patents are part of the same patent family. Plaintiff Webroot owns by assignment the entire right, title, and interest in and to the ’243 and ’932 Patents.

104. The ’243 Patent, entitled “Backwards Researching Activity Indicative of Pestware,” was filed on April 20, 2006 and was duly and legally issued by the USPTO on June 12, 2012. A true and correct copy of the ’243 Patent is attached as Exhibit 10.

105. The ’932 Patent, also entitled “Backwards Researching Activity Indicative of Pestware,” was filed on June 6, 2012 and was duly and legally issued by USPTO on May 6, 2014. The ’932 Patent claims priority to the ’243 Patent. A true and correct copy of the ’932 Patent is attached as Exhibit 12.

106. At the time the ’243 and ’932 Patents were filed, conventional pestware-detection-and-removal solutions struggled to stay ahead of evolving pestware because they relied on “definitions of known pestware to search for and remove files on a protected system.” (Exhibit 10, ’243 Patent, 1:63-65.) “Th[os]e definitions were often slow and cumbersome to create,” and “it was often difficult to locate the pestware in order to create the definitions,” in the first place. (*Id.* at 1:65-67.) Furthermore, relying on definitions made it difficult to distinguish wanted pestware from unwanted pestware. (*Id.* at 1:60-62.)

107. To overcome these and other shortcomings, the ’243 and ’932 Patents describe and claim novel “systems and methods for discovering the source of activity that is indicative of pestware.” (*Id*. at 2:17-19.) For example, the ’243 and ’932 Patents describe and claim the use of a kernel-mode driver to monitor the behavior of processes running on a computer, for example monitoring API calls, to detect pestware. (*Id.* at 4:24-29.) The inventors found that this approach provided technical advantages, such as the ability to “intercept” “when a process (*e.g.*, the pestware

process) attempt[ed] to spawn … another pestware process or alter a registry entry, … before it [wa]s carried out by an operating system of the protected computer." (*Id.* at 5:46-50.)

108. Moreover, the '243 and '932 Patents describe and claim techniques that enable the identification of suspected pestware as well as externally networked sources that, for example, may be the source of the pestware on the system. This approach provides benefits, such as identifying the source of malicious malware, which can be used to identify new malware (*e.g.*, based on a known source of malware) as well as to block potential malware based on a known malicious source.

<u>U.S. Patent No. 8,181,244</u>

109. The '244 Patent, entitled "Backwards Researching Activity Indicative of Pestware," was filed on April 20, 2006 and was duly and legally issued by USPTO on May 15, 2012. A true and correct copy of the '244 Patent is attached as Exhibit 9. Plaintiff Webroot owns by assignment the entire right, title, and interest in and to the '244 Patent.

110. At the time the '244 Patent was filed, prior-art pestware-detection-and-removal solutions struggled to stay ahead of evolving pestware. Prior-art pestware-detection-and-removal solutions relied on "definitions of known pestware to search for and remove files on a protected system." (Exhibit 9, '244 Patent, 1:63-65.) "Th[os]e definitions were often slow and cumbersome to create," and "it was often difficult to locate the pestware in order to create the definitions," in the first place. (*Id.* at 1:65-67.) Furthermore, relying on definitions made it difficult to distinguish wanted pestware from unwanted pestware. (*Id.* at 1:60-62.)

111. To overcome these shortcomings, the '244 Patent introduced novel "systems and methods for discovering the source of activity that is indicative of pestware." (*Id.* at 2:17-19.) The '244 Patent employed a kernel-mode driver to monitor the behavior of processes running on a

computer, for example monitoring API calls, to detect pestware. (*Id.* at 4:24-29.) Once the ’244 Patent detected “pestware activity,” the ’244 Patent issued a timestamp and assembled a log of events that occurred at or around that time. (*Id.* at 6:2-9.) Then, using the log of events, the ’244 Patent was able to “trac[e]…the pestware activity to an origin of the pestware…associated with the activity.” (*Id.* at 8:23-26.) Additionally, by using a kernel-mode driver, the ’244 Patent was able to “intercept” “when a process (*e.g.*, the pestware process) attempt[ed] to spawn…another pestware process or alter a registry entry, … before it [wa]s carried out by an operating system of the protected computer.” (*Id.* at 5:46-50.)

112. The ’244 Patent describes and claims technological solutions to a technological problem. For example, embodiments enable the identification of suspected pestware as well as externally networked sources that, for example, may be the source of the pestware on the system. This approach provides benefits, such as identifying the source of malicious malware, which can be used to identify new malware (*e.g.*, based on a known source of malware) as well as to block potential malware based on a known malicious source.

<u>ACCUSED PRODUCTS</u>

107. Defendant offers and sells the Accused Products including Sophos’ Intercept X Advanced with EDR and XDR, Sophos Web Appliance, Sophos XG Firewall, Sophos’ Synchronized Security, Sophos Endpoint Protection Advanced, as well as products and services with similar functionality. These products provide and implement malware detection, network security, and endpoint protection platforms for individuals and enterprises and incorporate Plaintiffs’ patented technologies.

108. Sophos’ Intercept X Advanced with XDR (“Intercept X”), formerly known as Intercept X Advanced with EDR, is an endpoint protection platform that establishes forensic attack

chains for infections and leverages machine learning techniques to detect malware.

Simplified events chain

At the top of every Sophos generated threat case (excludes Admin generated) you will see the simplified events chain. This gives you the very basic details of what happened.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html.)

109. Sophos' Intercept X also has an "SKU" called "Endpoint Protection Advanced" that includes a "Patch Assessment" module, whereby endpoints are scanned for whether they are missing "critical threat-related patches." (*See* https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/Sophos-Endpoint-dsna.pdf.)

> With our Patch Assessment in Endpoint we prioritize the most critical patches for you by tying them to the threats they prevent. Be confident in your security knowing that our patch assessment identifies, prioritizes and scans for critical threat-related patches. And it's integrated into our Endpoint Protection 10—delivered in one deployment and managed from a single console.

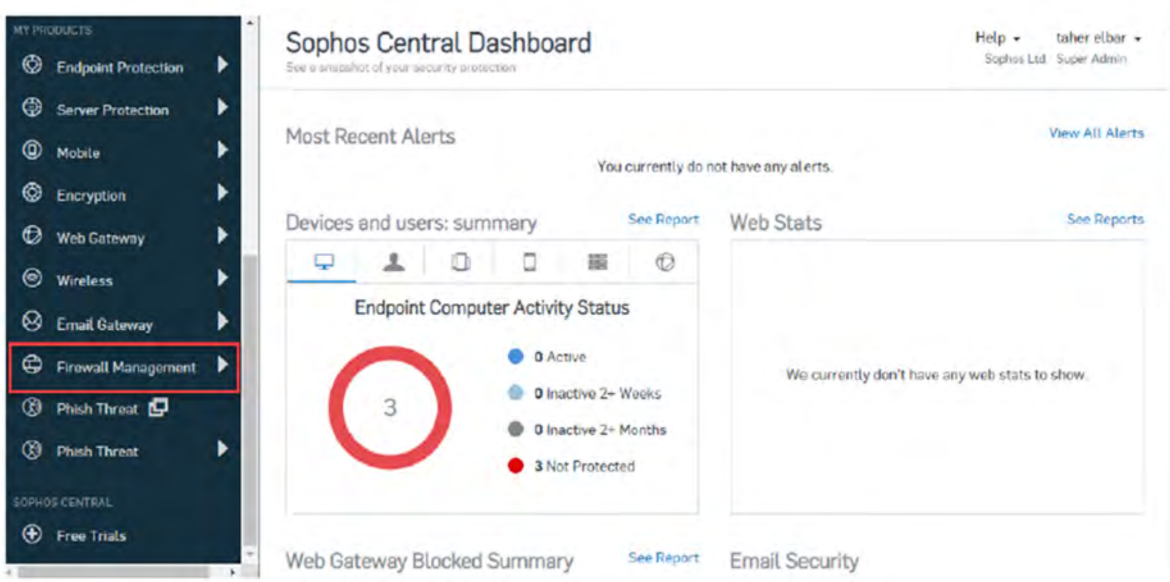
(*See* https://www.sophos.com/enus/medialibrary/PDFs/factsheets/sophosendpointpatchassessmentdsna.pdf.)

110. Sophos' Synchronized Security is an integrated cybersecurity platform that manages Sophos' endpoint security software products, such as XG Firewall and Intercept X, from a central point via a cloud-based Control Center. It receives "Security Heartbeat[s]" that share "health, security and security information" from Sophos' endpoint security products installed on various endpoints across a network protected by Synchronized Security.

111. The Sophos Synchronized Security Control Center's dashboard displays information about endpoints in the network. The details of this information include the name of the computer, IP address, operating system, the Sophos security products installed on the endpoint, and the health status of the endpoint.

With Synchronized Security, Sophos products share threat, health, and security information in real time via the Security Heartbeat™ and respond automatically to threats.

This quick start guide walks you through enabling Synchronized Security between Sophos XG Firewall and Sophos endpoint and server protection managed through Sophos Central (including Sophos Intercept X and Sophos Intercept X for Server).



Control center

The control center appears as soon as you sign in.

The control center provides a single screen snapshot of the state and health of the security system.

User & device insights panel

Security Heartbeat widget

Security Heartbeat widget provides the health status of all endpoint devices. An endpoint device is an internet-capable computer hardware device connected to Sophos XG Firewall via Sophos Central. The endpoint sends a heartbeat signal at regular intervals and also informs about potential threats to the Sophos XG Firewall.

(*See* WBR_SOP000239, https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophos-quick-start-guide-synchronized-security.pdf; *see also* https://docs.sophos.com/nsg/sophos-firewall/17.5/Help/en-us/webhelp/onlinehelp/nsg/sfos/concepts/SSLVPNLiveUsersManage.html.)

FIRST CAUSE OF ACTION
(INFRINGEMENT OF THE ’250 PATENT)

112. Plaintiffs reallege and incorporate by reference the allegations of the preceding paragraphs of this Complaint.

113. Defendant has infringed and continue to infringe one or more claims of the ’250 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. the Accused Products, including features such as Intercept X Advanced with XDR (“Intercept X”), at least when used for their ordinary and customary purposes, practice each element of at least claim 1 of the ’250 Patent as described below.

114. For example, claim 1 of the ’250 Patent recites:

> 1. A method of classifying a computer object as malware, the method comprising:
>
> at a base computer, receiving data about a computer object from each of plural remote computers on which the object or similar objects are stored, the data including information about the behaviour of the object running on one or more remote computers;

determining in the base computer whether the data about the computer object received from the plural computers indicates that the computer object is malware;

classifying the computer object as malware when the data indicates that the computer object is malware; when the determining does not indicate that the computer object is malware, initially classifying the computer object as not malware;

automatically generating a mask for the computer object that defines acceptable behaviour for the computer object, wherein the mask is generated in accordance with normal behaviour of the object determined from said received data;

running said object on at least one of the remote computers;

automatically monitoring operation of the object on the at least one of the remote computers;

allowing the computer object to continue to run when behaviour of the computer object is permitted by the mask:

disallowing the computer object to run when the actual monitored behaviour of the computer object extends beyond that permitted by the mask; and,

reclassifying the computer object as malware when the actual monitored behaviour extends beyond that permitted by the mask.

115. The Accused Products perform each element of the method of claim 1 of the ’250 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform *a method for classifying a computer object as malware*, as further explained below. For example, Intercept X with XDR (“Intercept X”) “scans across [the] entire environment and highlight[s] suspicious activity, anomalous behavior and other IT issues.” Intercept X displays “threats,” including processes classified as malware, in its “Sophos Central” and “Threat Analysis Center” dashboards.

XDR builds upon that solid foundation by adding even more data and context that both increases visibility and gives the user even more insight during an investigation. This results in faster and more accurate incident detection and response. Additional data sources can include firewall, email, cloud and mobile information. For example, adding in firewall data makes it simple to correlate a malicious traffic detection by the firewall with a compromised endpoint, or to see which application is causing the office network connection to run slowly.

One of the most valuable ways to use XDR is to begin with the 'macro' spotlight that gives you the tools to quickly scan across your entire environment and highlight suspicious activity, anomalous behavior and other IT issues. When an issue is identified you can then hone-in on a device of interest, pulling live data or remotely accessing the device in order to dig deeper and take remedial action.

(*See* WBR_SOP000172, https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/sophos-xdr-beginner-guide.pdf.)

116. The Accused Products perform a method that includes *receiving data at a base computer about a computer object from each of plural remote computers on which the object or similar objects are stored, the data including information about the behaviour of the object running on one or more remote computers*. For example, each endpoint on which Intercept X is installed sends data about the processes executing on it to the cloud-based "Sophos Central" computer which stores that data in a database and manages endpoints, which include remote computers, within a network. This data includes information about the behavior of an object running on one or more remote computers. For example, data can be queried from each endpoint using "Live Discover" SQL queries through the "Threat Analysis Dashboard," to detect, for example, processes (running objects) that have made "[u]nusual changes to the registry" or to "search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere." Data about each process is automatically analyzed, marked as a "threat case" if appropriate, and displayed as such in the "Threat Analysis Center." Moreover, such data is also periodically uploaded by each endpoint to a cloud-based computer "Data Lake," which can be queried, for example, to obtain data about which processes executed on a given endpoint.

Live Discover

Live Discover allows you to check the devices that Sophos Central is managing, look for signs of a threat, or assess compliance.

You can use Live Discover queries to search devices for signs of threats that haven't been detected by other Sophos features. For example:

- Unusual changes to the registry.
- Failed authentications.
- A process running that is very rarely run.

You can also search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere, or if a user reports suspicious behavior on their device.

You can also check the compliance of each device. For example, you can search for out-of-date software or browsers with insecure settings.

This page tells you how to use Live Discover. You can also familiarize yourself with it by completing the Sophos XDR Training.

How queries work

We provide a range of queries for you to use to check your devices. You can use them as they are, or edit them (you'll need to be familiar with osquery or SQL). You can also create queries.

You can run queries to get information from different sources:

- Endpoint queries get the latest information from devices that are currently connected.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/learningContents/LiveDiscover.html; *see also* WBR_SOP000529, https://community.sophos.com/intercept-x-endpoint/b/blog/posts/introducing-the-new-threat-analysis-center.)

Data Lake queries

Data Lake queries let you search security and compliance data that your devices upload to the cloud.

You can run Data Lake queries with Live Discover, a feature in our Threat Analysis Center.

Live Discover now lets you choose which data source you use when you set up and run a query:

- Endpoints that are currently connected.
- The Data Lake in the cloud.

For help with Live Discover *See* Live Discover.

How the Data Lake works

We host the Data Lake and provide scheduled "hydration queries" that define which data your endpoints upload to it.

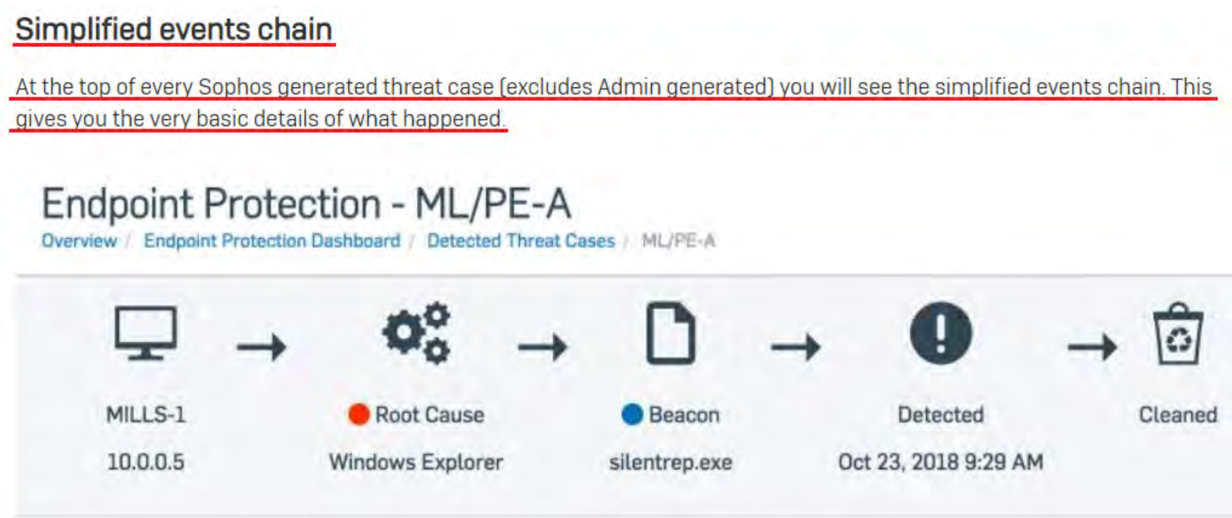
However, before you use Data Lake queries, you must make sure that data is being uploaded. To turn on uploads of data, see Data Lake uploads.

We store the data for 30 days.

We provide pre-prepared Data Lake queries you can run. You can use them as they are or edit them. You can also create your own queries.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/DataLakeQueries.html.)

117. The Accused Products perform a method that includes *determining in the base computer whether the data about the computer object received from the plural computers indicates that the computer object is malware*. For example, each endpoint on which Intercept X is installed sends data about the processes executing on it to the cloud-based "Sophos Central" computer, which stores that data in a database and manages endpoints (including remote computers), within a network. This data includes information about the behavior of an object running on one or more remote computers. For example, data can be queried from each endpoint using "Live Discover" SQL queries through the "Threat Analysis Dashboard," to detect, for example, processes (running objects) that have made "[u]nusual changes to the registry" or to "search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere." Data about each process is automatically analyzed, marked as a "threat case" if appropriate, and displayed as such in the "Threat Analysis Center." Moreover, such data is also periodically uploaded by each endpoint to a cloud-based computer "Data Lake," which can be queried, for example, to obtain data about which processes executed on a given endpoint.



(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl.)

118. The Accused Products perform a method that includes *classifying the computer object as malware when the data indicates that the computer object is malware; when the determining does not indicate that the computer object is malware, initially classifying the computer object as not malware*. For example, as explained above, the Accused Products use the data on processes that each endpoint sends to Sophos Central to determine whether those processes constitute malware. In the example below, Sophos Central has identified that the process "silentrep.exe" is a "threat case" and a variant of the malware class "ML/PE-A." When the data that Sophos Central receives does not indicate that the process is malware or potentially malicious, that process is not initially marked as a "threat case."

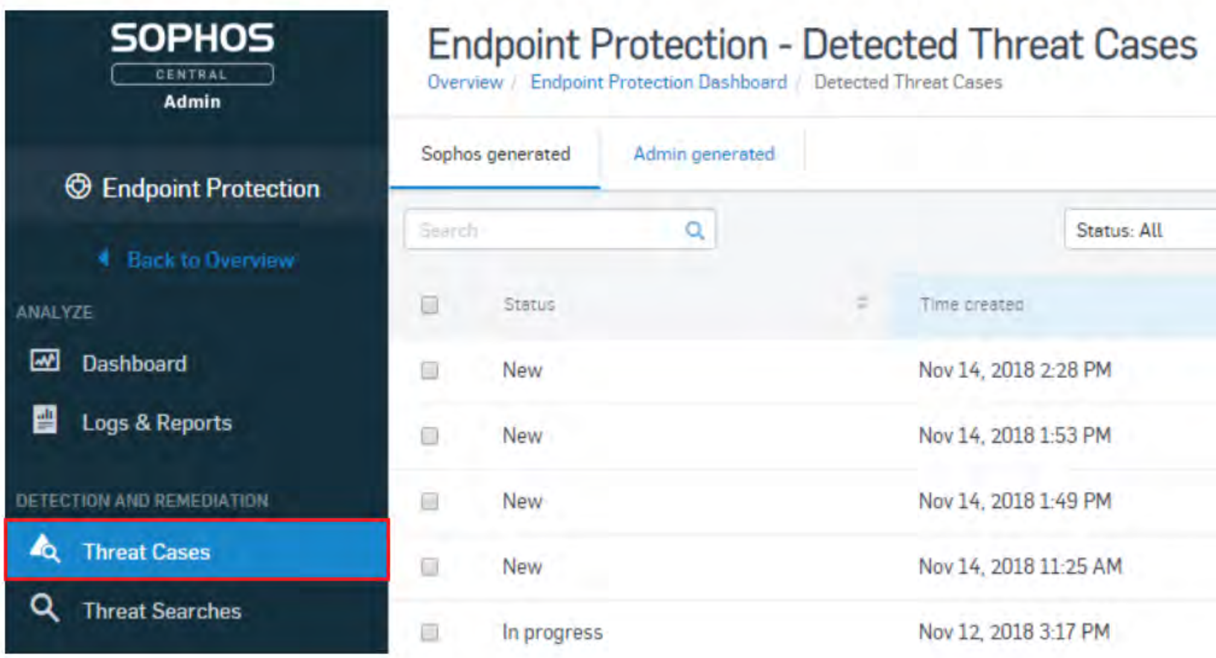
Simplified events chain

At the top of every Sophos generated threat case (excludes Admin generated) you will see the simplified events chain. This gives you the very basic details of what happened.

(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl; *see also* WBR_SOP000529, https://community.sophos.com/intercept-x-endpoint/b/ blog/posts/introducing-the-new-threat-analysis-center.)

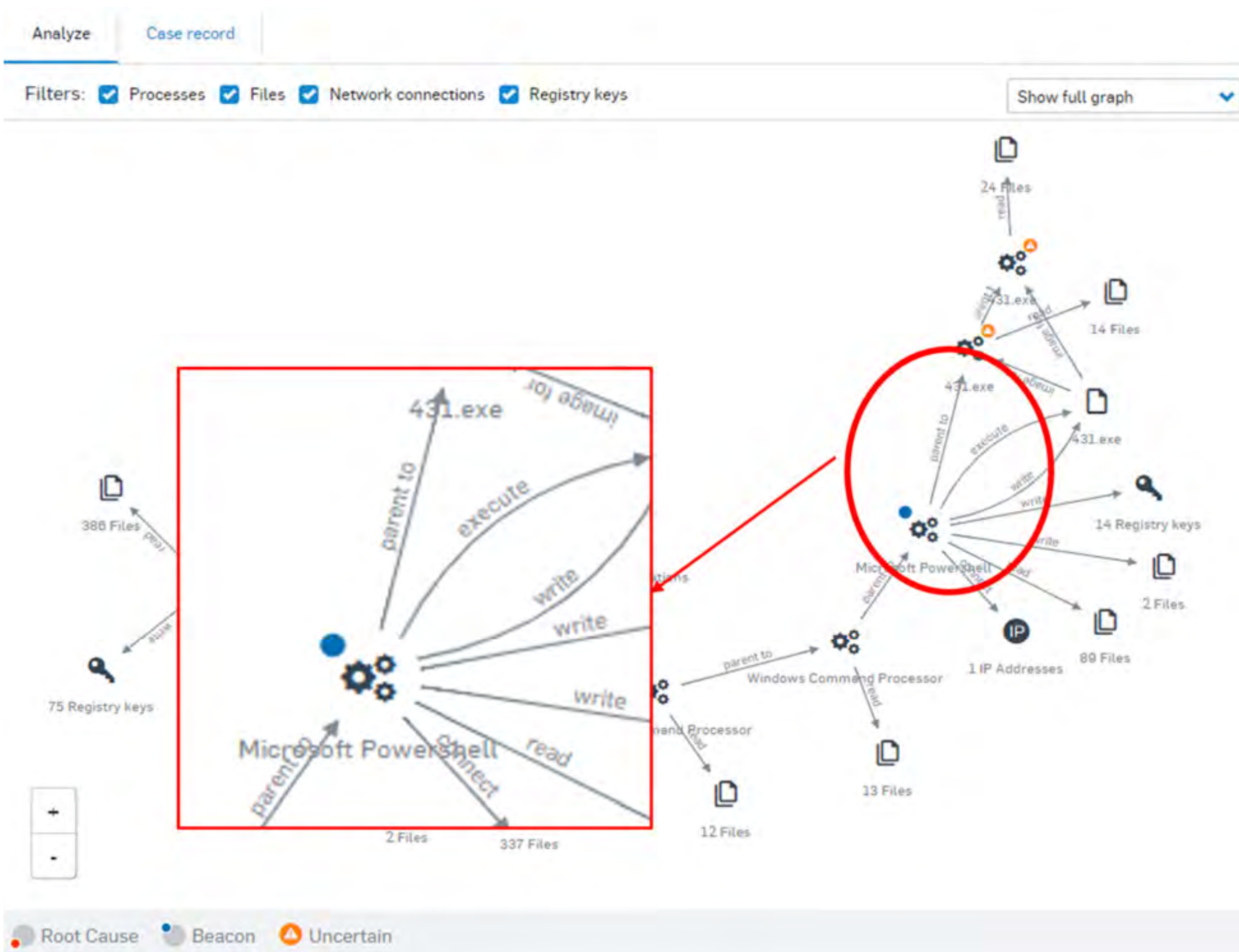
For customers with Sophos EDR, the full list of Threat Cases can be found in the below locations:

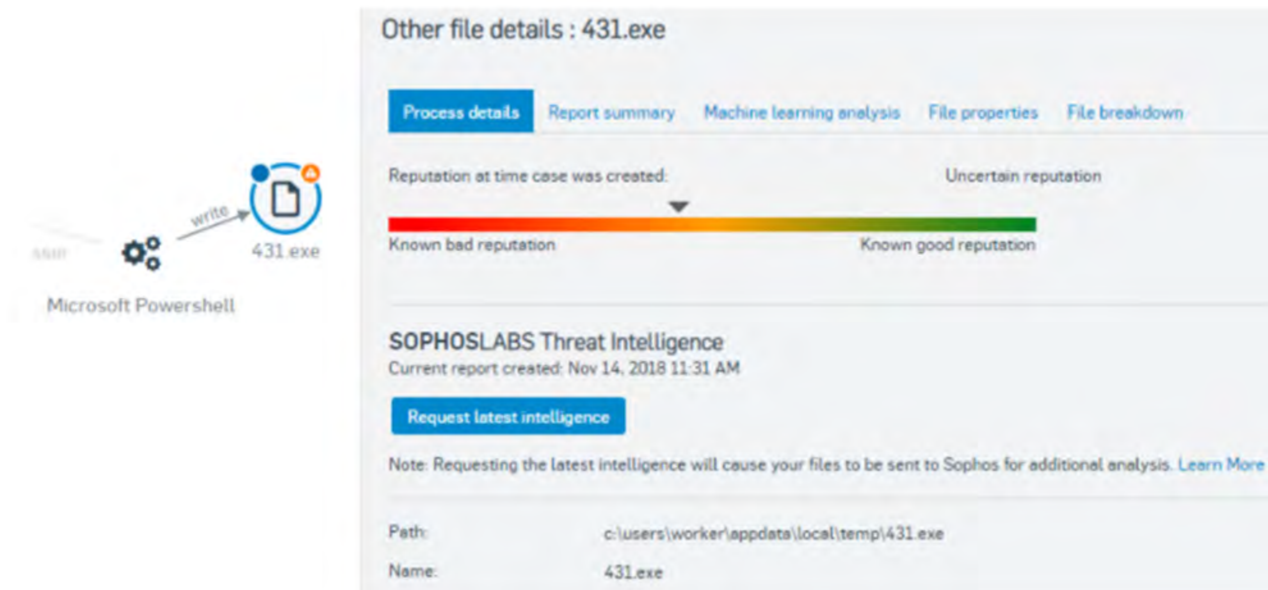
- Endpoint Protection > Detection and Remediation > Threat Cases
- Server Protection > Analyze > Threat Cases



To view a Threat Case click on the detection **Name**:

119. The Accused Products perform a method that includes *automatically generating a mask for the computer object that defines acceptable behavior for the computer object, wherein the mask is generated in accordance with normal behavior of the object determined from said received data*. For example, the Accused Products assess "threat cases" by detecting illicit behaviors (*e.g.*, behaviors that are not normal for the object) such as the modification of "registry keys." In particular, in the example shown below, the "Analyze" tab of a "threat case" displayed in the "Threat Analytics Center" illustrates the illicit execution, on an infected endpoint device, of the suspicious process "431.exe" by Microsoft Powershell, which is marked as the "Beacon," and the subsequent modification of "registry keys" by the file "431.exe" on the infected endpoint. The process "431.exe" is quarantined after its behavior is detected as being malicious.





Important: Customers who are **not** using Sophos XDR, you will only see the one **Process details** tab.

For customers using Sophos EDR, by pressing the **Request latest intelligence** button, the file will be retrieved out of the Sophos quarantine and submitted to SophosLabs. A couple of minutes later the four other tabs (Report summary, Machine learning analysis, File properties, File breakdown) pictured will be displayed. The purpose of these these additional tabs is to help display the various properties of the file in a simple way. This can be useful for various reasons, one of them is to feel confident that the file is indeed malicious and not something you want in your environment. For more information on SophosLabs Threat Intelligence, please see: Sophos Central: Threat intelligence overview.

(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl.)

120. Based on the data that it received about the behavior of processes (*e.g.*, "431.exe" and the behavior of the process "Microsoft PowerShell," which executed "431.exe"), Sophos Central determined that PowerShell exceeded the scope of normal behavior by being launched "with an obfuscated and very suspicious command line" and that "431.exe" illicitly modified registry keys and was written to "the users AppData location," which is "typically meant for data not executable," resulting in its quarantine. As such, Sophos Central defines what is typical or normal behavior for both malware and non-malware. For example, it explains that "[o]bfuscation is very typical in malicious code and is designed to hide the true goal behind the code." In another example, when one command shell launches anther command shell, that behavior is deemed

suspicious, meaning that when one command shell does not invoke another, it is deemed non-malware behavior: "We can also see that when CMD was launched it then launched another copy of CMD, this one with a similar suspicious command line."

(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl.)

121. The Accused Products perform a method that includes *running said object on at least one of the remote computers and automatically monitoring operation of the object on the at least one of the remote computers*. As explained above, the Accused Products identify "threat cases" as they occur on each endpoint. In particular, in the example shown above, the "Analyze"

tab of a "threat case" displayed in the "Threat Analytics Center" illustrates the illicit execution, on an infected endpoint device, of the suspicious process "431.exe" by Microsoft Powershell, which is marked as the "Beacon," and the subsequent modification of "registry keys" by the file "431.exe" on the infected endpoint. The process "431.exe" was initially allowed to execute until it suspiciously modified registry keys, at which point it was classified as malware and quarantined.

122. The Accused Products perform a method that includes *allowing the computer object to continue to run when behaviour of the computer object is permitted by the mask and disallowing the computer object to run when the actual monitored behaviour of the computer object extends beyond that permitted by the mask*. As explained and shown in the example above, the "Analyze" tab of a "threat case" displayed in the "Threat Analytics Center" illustrates the illicit execution, on an infected endpoint device, of the suspicious process "431.exe" by Microsoft Powershell, which is marked as the "Beacon," and the subsequent modification of "registry keys" by the file "431.exe" on the infected endpoint. The process "431.exe" was allowed to execute until it suspiciously modified registry keys, at which point it was quarantined.

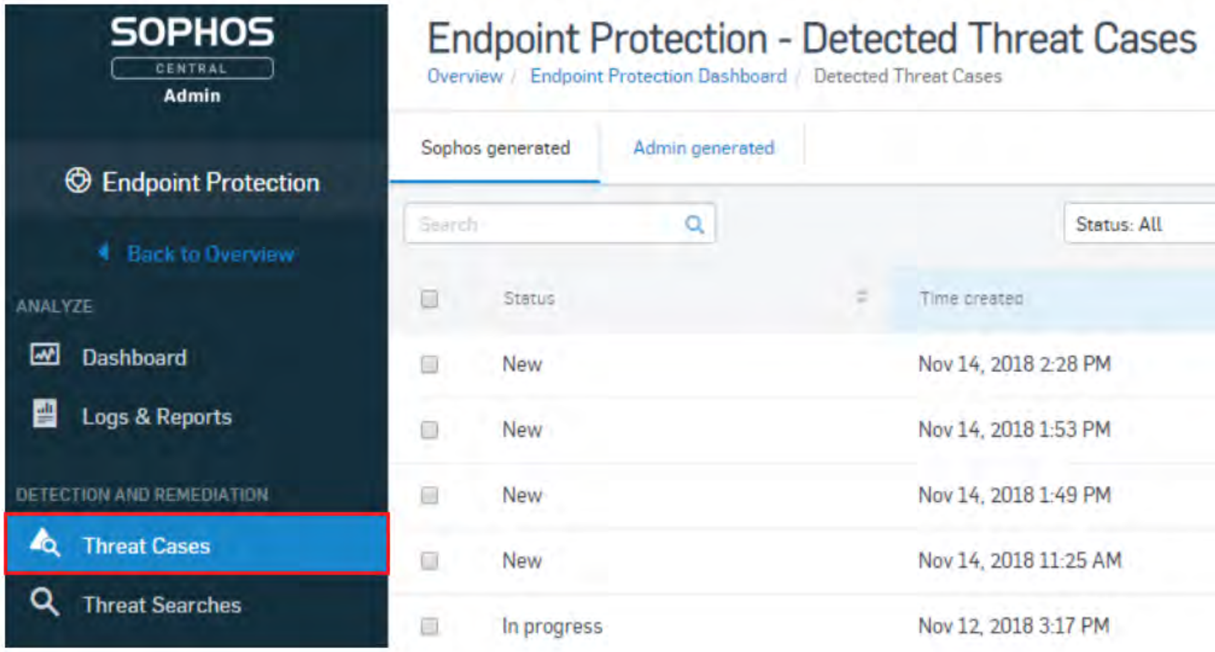
123. The Accused Products perform a method that includes *reclassifying the computer object as malware when the actual monitored behaviour extends beyond that permitted by the mask*. As explained above, and as shown in the example above, the "Analyze" tab of a "threat case" displayed in the "Threat Analytics Center" illustrates the illicit execution, on an infected endpoint device, of the suspicious process "431.exe" by Microsoft Powershell, which is marked as the "Beacon," and the subsequent modification of "registry keys" by the file "431.exe" on the infected endpoint. The process "431.exe" was initially allowed to execute until it suspiciously modified registry keys, at which point it was classified as malware and quarantined.

124. In another example, the Accused Products detected that the process "silentrep.exe"

was a variant of the malware class “ML/PE-A” and re-classified it as such.

For customers with Sophos EDR, the full list of Threat Cases can be found in the below locations:

- Endpoint Protection > Detection and Remediation > Threat Cases
- Server Protection > Analyze > Threat Cases



To view a Threat Case click on the detection **Name**:

Simplified events chain

At the top of every Sophos generated threat case (excludes Admin generated) you will see the simplified events chain. This gives you the very basic details of what happened.

(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl.)

125. Each claim in the ’250 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ’250 Patent.

126. Defendant has been aware of the ’250 Patent since at least the filing of this Complaint. Further, Plaintiffs have marked their products with the ’250 Patent, including on its web site, since at least July 2020.

127. Defendant directly infringes at least claim 1 of the ’250 Patent, either literally or under the doctrine of equivalents, by performing the steps described above. For example, on information and belief, Defendant performs the claimed method in an infringing manner as described above by running this software and system to protect its own computer and network operations. On information and belief, Defendant also performs the claimed method in an infringing manner when testing the operation of the Accused Products’ and corresponding systems. As another example, Defendant performs the claimed method when providing or administering services to third parties, customers, and partners using the Accused Products.

128. Defendant’s partners, customers, and end users of its Accused Products and corresponding systems and services directly infringe at least claim 1 of the ’250 Patent, literally or under the doctrine of equivalents, at least by using the Accused Products and corresponding systems and services, as described above.

129. Defendant has actively induced and is actively inducing infringement of at least claim 1 of the ’250 Patent with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b). For example, Defendant encourages and induces customers with specific intent to use Sophos’ security software in a manner that infringes claim 1 of the ’250 Patent at least by offering and providing software that performs a method that infringes claim 1 when installed and operated by the customer, and by

engaging in activities relating to selling, marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including the activities described below.

130. Defendant encourages, instructs, directs, and/or requires third parties—including their certified partners and/or customers—to perform the claimed method using the software, services, and systems in infringing ways, as described above.

131. Defendant further encourages and induces its customers to infringe claim 1 of the ’250 Patent: 1) by making its security services available on its website, providing applications that allow users to access those services, widely advertising those services, and providing technical support and instructions to users, and 2) through activities relating to marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including their Sophos security software, and services in the United States. (*See* WBR_SOP000506, https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx; WBR_SOP000525, https://partners.sophos.com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1; WBR_SOP000209, https://secure2.sophos.com/en-us/security-news-trends/whitepapers/gated-wp/cybersecurity-system-buyers-guide.aspx%23formFrame; *see also* WBR_SOP000214, https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophos-synchronized-security-ds.pdf.)

132. For example, on information and belief, Defendant shares instructions, guides, and manuals, which advertise and instruct third parties on how to use the software as described above, including at least customers and partners. (*Id*.) On further information and belief, Defendant also provides customer service and technical support to purchasers of the Accused Products and corresponding systems and services, which directs and encourages customers to perform certain actions that use the Accused Products in an infringing manner. (*Id.*)

133. Defendant and/or its partners recommend and sell the Accused Products and provide technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Defendant and/or one of its partners, which obligates each customer to perform certain actions in order to use the Accused Products. Further, in order to receive the benefit of Defendant and/or its partner's continued technical support and their specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that infringes the '250 Patent.

134. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Product remains operational for each customer through ongoing technical support, on information and belief, Defendant and/or its partners affirmatively aid and abet each customer's use of the Accused Products in a manner that performs the claimed method of, and infringes, the '250 Patent.

135. Defendant also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the inventions claimed, and not a staple article or commodity of commerce suitable for substantial non-infringing uses.

136. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the behavioral "threat analysis" and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. For example, without compiling and

analyzing data about the behavior of an object running on one or more remote computers, the Accused Products could not detect processes (running objects) that have made "[u]nusual changes to the registry" or to "search devices for signs of a suspected or known threat." (*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/learningContents/ LiveDiscover.html). These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the '250 Patent, that functionality could not be performed.

137. Additionally, the accused functionality, including the behavioral threat analysis and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without a "Threat Analysis Center" that compiles and correlates behavioral data from processes that run on the endpoints protected by the Accused Products, the Accused Products could not assess which behaviors on any given endpoint constitute a threat. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the '250 Patent, that functionality could not be performed.

138. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Defendant constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and

systems. For example, the Accused Products and accused functionalities (including the behavioral threat analysis functionality) constitute a material part of the inventions claimed because such analysis is integral to the processes identified above (such as to generating a "mask for the computer object that defines acceptable behaviour for the computer object") as recited in the claims of the '250 Patent. None of these products are staple goods—they are sophisticated and customized cyber security and malware detection products, methods, and systems.

139. Defendant's infringing actions have continued after the original Complaint was filed. Defendant had knowledge of the '250 Patent and of the specific conduct that constitutes infringement of the '250 Patent at least based on the original Complaint, but has continued to engage in the infringing activity, including by selling, making, configuring, and installing the Accused Products and performing the accused functionality, and by engaging in activities that constitute inducing infringement and contributory infringement as described above.

140. On information and belief, the infringing actions of each partner, customer, and/or end-user of the Accused Products are attributable to Defendant. For example, on information and belief, Defendant directs and controls the activities or actions of its partners or others in connection with the Accused Products by contractual agreement or otherwise requiring partners or others to provide information and instructions to customers who acquire the Accused Products which, when followed, results in infringement. Defendant further directs and controls the operation of devices executing the Accused Products by programming the software which, when executed by a customer or end user, perform the claimed method of at least claim 1 of the '250 Patent.

141. Plaintiffs have suffered and continue to suffer damages, including lost profits, as a result of Defendant's infringement of the '250 Patent. Defendant is therefore liable to Plaintiffs under 35 U.S.C. § 284 for damages in an amount that adequately compensates Plaintiffs for

Defendant's infringement, but no less than a reasonable royalty.

142. Plaintiffs will continue to suffer irreparable harm unless this Court preliminarily and permanently enjoins Defendant, its agents, employees, representatives, and all others acting in concert with Defendant from infringing the '250 Patent.

143. Defendant's infringement of the '250 Patent is knowing and willful. Defendant had actual knowledge of the '250 Patent at least by the time Plaintiffs filed this lawsuit and had constructive knowledge of the '250 Patent from at least the date Plaintiffs marked their products with the '250 Patent and/or provided notice of the '250 Patent on their website.

144. On information and belief, despite Defendant's knowledge of the Asserted Patents and Plaintiffs' patented technology, Defendant made the deliberate decision to sell products and services that they knew infringe these patents. Defendant's continued infringement of the '250 Patent with knowledge of the '250 Patent constitutes willful infringement.

145. Plaintiffs' allegations of infringement, indirect infringement, and willful infringement with respect to this patent are further set forth in Plaintiffs' Disclosure of Asserted Claims and Preliminary Infringement Contentions Pursuant to the Court's Standing Order Governing Patent Proceedings served July 12, 2022.

SECOND CAUSE OF ACTION
(INFRINGEMENT OF THE '389 PATENT)

146. Plaintiffs reallege and incorporate by reference the allegations of the preceding paragraphs of this Complaint.

147. Sophos has infringed and continues to infringe one or more claims of the '389 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including features such as Intercept X Advanced with XDR ("Intercept X with XDR"), at least when used for their ordinary

and customary purposes, practice each element of at least claim 1 of the ’389 Patent as described below.

148. For example, claim 1 of the ’389 Patent recites:

> 1. A method of classifying a computer object as malware, the method comprising:
>
> at a base computer, receiving data about a computer object from a first remote computer on which the computer object or similar computer objects are stored, wherein said data includes information about events initiated or involving the computer object when the computer object is created, configured or runs on the first remote computer, said information including at least an identity of an object initiating the event, the event type, and an identity of an object or other entity on which the event is being performed;
>
> at the base computer, receiving data about the computer object from a second remote computer on which the computer object or similar computer objects are stored, wherein said data includes information about events initiated or involving the computer object when the computer object is created, configured, or runs on the second remote computer, said information including at least an identity of an object initiating the event, the event type, and an identity of an object or other entity on which the event is being performed;
>
> storing, at the base computer, said data received from the first and second remote computers;
>
> correlating, by the base computer, at least a portion of the data about the computer object received from the first remote computer to at least a portion of the data about the computer object received from the second remote computer;
>
> comparing, by the base computer, the correlated data about the computer object received from the first and second remote computers to other objects or entities to identify relationships between the correlated data and the other objects or entities; and
>
> classifying, by the base computer, the computer object as malware on the basis of said comparison.

149. The Accused Products perform each of the method steps of claim 1 of the ’389 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform a *method of classifying a computer object as malware*, as further explained below. For example,

Intercept X with XDR ("Intercept X") "scans across [the] entire environment and highlight[s] suspicious activity, anomalous behavior and other IT issues." Intercept X displays "threats," including processes classified as malware, in its "Sophos Central" and "Threat Analysis Center" dashboards.

> XDR builds upon that solid foundation by adding even more data and context that both increases visibility and gives the user even more insight during an investigation. This results in faster and more accurate incident detection and response. Additional data sources can include firewall, email, cloud and mobile information. For example, adding in firewall data makes it simple to correlate a malicious traffic detection by the firewall with a compromised endpoint, or to see which application is causing the office network connection to run slowly.
>
> One of the most valuable ways to use XDR is to begin with the 'macro' spotlight that gives you the tools to quickly scan across your entire environment and highlight suspicious activity, anomalous behavior and other IT issues. When an issue is identified you can then hone-in on a device of interest, pulling live data or remotely accessing the device in order to dig deeper and take remedial action.

(*See* WBR_SOP000172, https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/sophos-xdr-beginner-guide.pdf; *see also* WBR_SOP000529, https://community.sophos.com/intercept-x-endpoint/b/blog/posts/introducing-the-new-threat-analysis-center.)

150. The Accused Products perform a method that includes *at a base computer, receiving data about a computer object from a first remote computer on which the computer object or similar computer objects are stored*. For example, each endpoint on which Intercept X is installed sends data about the processes executing on it to the cloud-based "Sophos Central," which stores that data in a database and manages endpoints within a network. For example, data can be queried from each endpoint using "Live Discover" SQL queries through the "Threat Analysis Dashboard," to detect, for example, processes that have made "[u]nusual changes to the registry" or to "search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere." Data about each process is automatically analyzed, marked as a "threat case," and displayed as such in the "Threat Analysis Center." Moreover, such data is also periodically uploaded by each endpoint to a cloud-based "Data Lake," which can be queried, for example, to

obtain data about which processes executed on a given endpoint when it is offline.

Live Discover

Live Discover allows you to check the devices that Sophos Central is managing, look for signs of a threat, or assess compliance.

You can use Live Discover queries to search devices for signs of threats that haven't been detected by other Sophos features. For example:

- Unusual changes to the registry.
- Failed authentications.
- A process running that is very rarely run.

You can also search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere, or if a user reports suspicious behavior on their device.

You can also check the compliance of each device. For example, you can search for out-of-date software or browsers with insecure settings.

This page tells you how to use Live Discover. You can also familiarize yourself with it by completing the Sophos XDR Training.

How queries work

We provide a range of queries for you to use to check your devices. You can use them as they are, or edit them (you'll need to be familiar with osquery or SQL). You can also create queries.

You can run queries to get information from different sources:

- Endpoint queries get the latest information from devices that are currently connected.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/learningContents/LiveDiscover.html; *see also* WBR_SOP000529, https://community.sophos.com/intercept-x-endpoint/b/blog/posts/introducing-the-new-threat-analysis-center.)

Data Lake queries

Data Lake queries let you search security and compliance data that your devices upload to the cloud.

You can run Data Lake queries with Live Discover, a feature in our Threat Analysis Center.

Live Discover now lets you choose which data source you use when you set up and run a query:

- Endpoints that are currently connected.
- The Data Lake in the cloud.

For help with Live Discover see Live Discover.

How the Data Lake works

We host the Data Lake and provide scheduled "hydration queries" that define which data your endpoints upload to it.

However, before you use Data Lake queries, you must make sure that data is being uploaded. To turn on uploads of data, see Data Lake uploads.

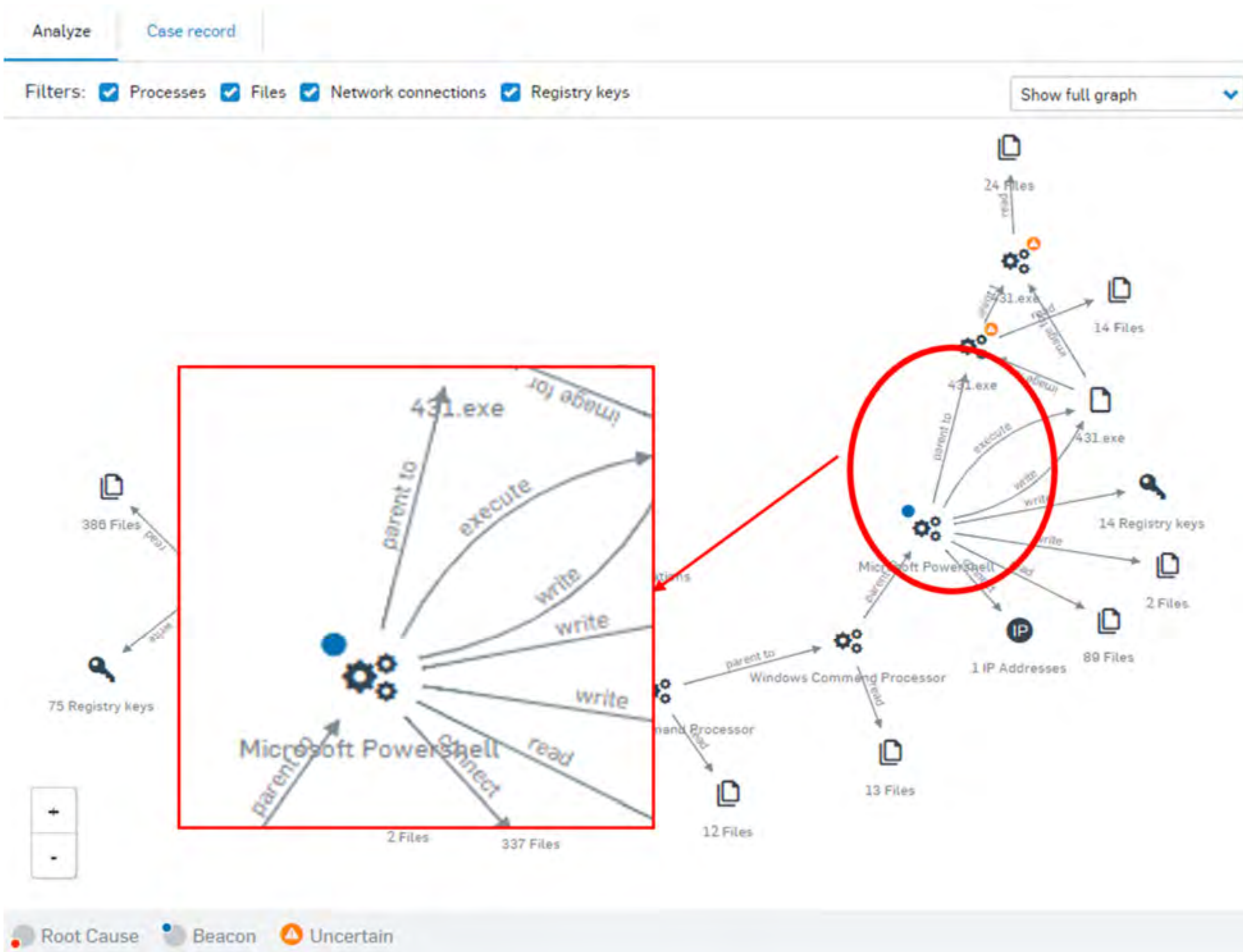
We store the data for 30 days.

We provide pre-prepared Data Lake queries you can run. You can use them as they are or edit them. You can also create your own queries.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/DataLakeQueries.html.)

151. The Accused Products perform a method that includes *wherein the data received from a first remote computer about a computer object includes information about events initiated or involving the computer object when the computer object is created, configured or runs on the first remote computer, said information including at least an identity of an object initiating the event, the event type, and an identity of an object or other entity on which the event is being performed*. As shown above, each endpoint on which Intercept X is installed sends data about the processes executing on it to the cloud-based "Sophos Central," which stores that data in a database and manages endpoints within a network.

152. As further evidence, the event data sent to Sophos Central by each endpoint includes event data generated when the file or process is created, configured or executed. The event data also includes incident details that describe the identity of objects and entities on which each event is performed. In particular, in the example shown below, the "Analyze" tab of a "threat case" displayed in the "Threat Analytics Center" illustrates the illicit execution, on an infected endpoint device, of the suspicious process "431.exe" by Microsoft Powershell, which is marked as the "Beacon," and the subsequent modification of "registry keys" by the file "431.exe" on the infected endpoint.



(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl.)

153. The Accused Products perform a method that includes wherein *data about the computer object from a second remote computer on which the computer object or similar computer objects are stored, wherein said data includes information about events initiated or involving the computer object when the computer object is created, configured, or runs on the second remote computer, said information including at least an identity of an object initiating the event, the event type, and an identity of an object or other entity on which the event is being performed.*

154. For example, as explained above, each endpoint on which Intercept X is installed sends data about the processes executing on it to the cloud-based "Sophos Central," which stores

that data in a database and manages endpoints within a network. For example, data can be queried using "Live Discover" SQL queries through the "Threat Analysis Dashboard," to detect, *e.g.*, processes that have made "[u]nusual changes to the registry" or to "search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere." Data about each process is automatically analyzed, marked as a "threat case," and displayed as such in the "Threat Analysis Center." Moreover, such data is also periodically uploaded by each endpoint to a cloud-based "Data Lake," which can be queried, for example, to obtain data about which processes executed on a given endpoint when it is offline. The Accused Products protect endpoints associated with "more than 500,000 organizations and millions of consumers in more than 150 countries." (*See* https://www.sophos.com/en-us/press-office/press-releases/2020/11/sophos-intercept-x-named-best-endpoint-security-solution.aspx.)

Live Discover

Live Discover allows you to check the devices that Sophos Central is managing, look for signs of a threat, or assess compliance.

You can use Live Discover queries to search devices for signs of threats that haven't been detected by other Sophos features. For example:

- Unusual changes to the registry.
- Failed authentications.
- A process running that is very rarely run.

You can also search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere, or if a user reports suspicious behavior on their device.

You can also check the compliance of each device. For example, you can search for out-of-date software or browsers with insecure settings.

This page tells you how to use Live Discover. You can also familiarize yourself with it by completing the Sophos XDR Training.

How queries work

We provide a range of queries for you to use to check your devices. You can use them as they are, or edit them (you'll need to be familiar with osquery or SQL). You can also create queries.

You can run queries to get information from different sources:

- Endpoint queries get the latest information from devices that are currently connected.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/learningContents/LiveDiscover.html; *see also* WBR_SOP000529, https://community.sophos.com/intercept-x-endpoint/b/blog/posts/introducing-the-new-threat-analysis-center; https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/DataLake Queries.html.)

155. As further evidence, the event data sent to Sophos Central by each endpoint includes event data generated when the file or process is created, configured and executed. The event data also includes incident details that describe the identity of objects and entities on which each event is performed. In particular, in the example shown above, the "Analyze" tab of a "threat case" displayed in the "Threat Analytics Center" illustrates the illicit execution, on an infected endpoint device, of the suspicious process "431.exe" by Microsoft Powershell, which is marked as the "Beacon," and the subsequent modification of "registry keys" by the file "431.exe" on the infected endpoint.

156. The Accused Products perform a method that includes *storing, at the base computer, said data received from the first and second remote computers*. For example, as explained above, Sophos Central stores data received from every endpoint and organizes it in a database. As another example, data from endpoints is also organized into a "Data Lake," which can be queried, for example, to obtain data about which processes executed on a given endpoint when it is offline.

157. Sophos Central stores data from remote computers in a centralized "Data Lake."

Data Lake storage limits

There are limits on how much data you can store in the Sophos Data Lake.

For devices we set the limits as follows:

- A daily limit for one device.
- A monthly limit for all your devices.

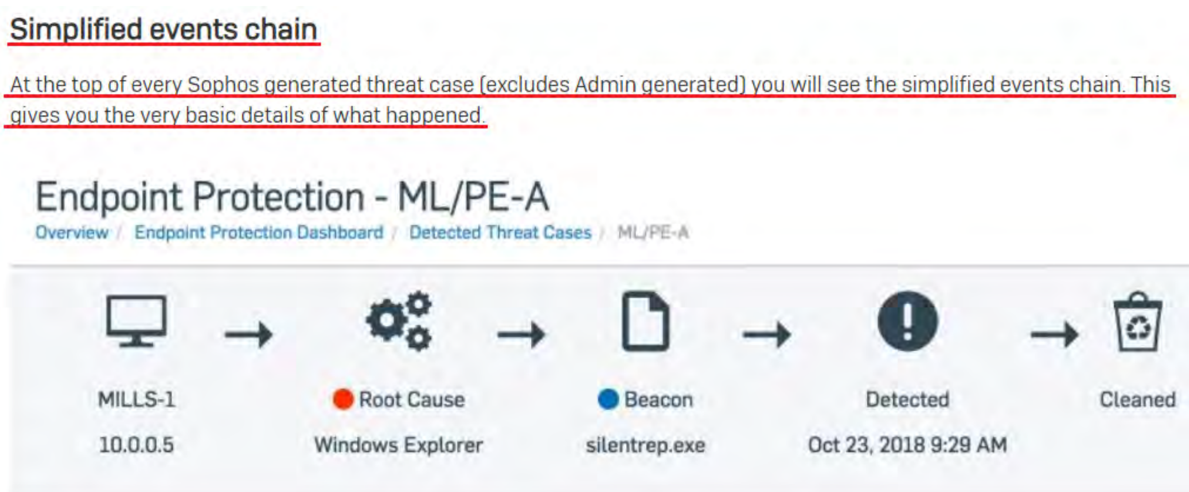
For cloud assets we set limits as described in the Sophos Cloud Optix section.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/DataLakeStorageLimits.html.)

158. The Accused Products perform a method that includes *correlating, by the base computer, at least a portion of the data about the computer object received from the first remote computer to at least a portion of the data about the computer object received from the second remote computer*. For example, each endpoint on which Intercept X is installed sends data about the processes executing on it to the cloud-based "Sophos Central," which stores and organizes that data in a database and manages endpoints within a network. Data about those processes, such as which actions of events they have initiated on their endpoints, and which other endpoints also ran processes initiating such actions, are correlated within that database, and can be queried on the basis of those correlations to, for example, "search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere."

159. As another example, by using "Live Discover" SQL queries to a correlated database through the "Threat Analysis Dashboard," a system administrator can obtain a list of processes, across all connected endpoints, that have made certain "[u]nusual changes to the registry" or "Failed Authentications" in a particular way. (*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/learningContents/LiveDiscover.html.)

160. The Accused Products perform a method that includes *comparing, by the base computer, the correlated data about the computer object received from the first and second remote computers to other objects or entities to identify relationships between the correlated data and the other objects or entities.* As explained above, the Accused Products use the data on processes that each endpoint sends to Sophos Central to identify relationships between those processes and malware to identify "threat cases." These comparisons allow the Accused Products to search for signs of a suspected or known threat if Sophos Central has found the threat elsewhere. In the example below, Intercept X has identified that the process "silentrep.exe" is a variant of the malware "ML/PE-A."



(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl.)

161. The Accused Products perform a method that includes *classifying, by the base computer, the computer object as malware based on said comparison*. For example, as explained above, Intercept X identified that the data Sophos Central received about the process "silentrep.exe" indicated that it was a variant of malware "ML/PE-A" and on the basis of that comparison, classified it as malware.

162. Each claim in the ’389 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ’389 Patent.

163. Sophos has been aware of the ’389 Patent since at least the filing of this Complaint. Further, Plaintiffs have marked its products with the ’389 Patent, including on its web site, since at least July 2020.

164. Defendant directly infringes at least claim 1 of the ’389 Patent, literally or under the doctrine of equivalents, by performing the steps described above. For example, on information and belief, Defendant performs the claimed method in an infringing manner as described above by running this software and corresponding systems to protect its own computer and network operations. On information and belief, Defendant also performs the claimed method as described above when testing the operation of the Accused Products and corresponding systems. As another example, Defendant performs the claimed method as described above when providing or administering services to third parties, customers, and partners using the Accused Products.

165. Defendant’s partners, customers, and end users of the Accused Products and corresponding systems and services directly infringe at least claim 1 of the ’389 Patent, literally or under the doctrine of equivalents, at least by using the Accused Products and corresponding systems and services, as described above.

166. Defendant has actively induced and are actively inducing infringement of at least claim 1 of the ’389 Patent with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b). For example, Defendant encourages and induces customers with specific intent to use Sophos’s security software in a manner that infringes claim 1 of the ’389 Patent at least by offering and providing software that performs a method that infringes claim 1 when installed and operated by the

customer, and by engaging in activities relating to selling marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including the activities described below.

167. Defendant encourages, instruct, direct, and/or require third parties—including its certified partners and/or customers—to perform the claimed method using the software, services, and systems in infringing ways, as described above.

168. Defendant further encourages and induces its customers to infringe claim 1 of the ’389 Patent: 1) by making its security services available on its website, providing applications that allow users to access those services, widely advertising those services, and providing technical support and instructions to users, and 2) through activities relating to marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including its Sophos security software and services in the United States. (*See* https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx; WBR_SOP000525, https://partners.sophos.com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1; WBR_SOP000209, https://secure2.sophos.com/en-us/security-news-trends/whitepapers/gated-wp/cybersecurity-system-buyers-guide.aspx%23formFrame; *see also* WBR_SOP000214, https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophos-synchronized-security-ds.pdf.)

169. For example, on information and belief, Defendant shares instructions, guides, and manuals, which advertise and instruct third parties on how to use the software as described above, including at least customers and partners. (*Id.*) On further information and belief, Defendant also provides customer service and technical support to purchasers of the Accused Products and corresponding system and services, which directs and encourages customers to perform certain

actions that use the Accused Products in an infringing manner. (*Id*.)

170. Defendant and/or its partners recommend and sell the Accused Products and provide technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Defendant and/or one of its partners, which obligates each customer to perform certain actions in order to use the Accused Products. Further, in order to receive the benefit of Defendant and/or its partner's continued technical support and their specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that infringes the '389 Patent.

171. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Product remains operational for each customer through ongoing technical support, on information and belief, Defendant and/or its partners affirmatively aid and abet each customer's use of the Accused Products in a manner that performs the claimed method of, and infringes, the '389 Patent.

172. Defendant also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the claimed methods, and not a staple article or commodity of commerce suitable for substantial non-infringing uses. Indeed, as shown above, the Accused Products and the example functionality described below have no substantial non-infringing uses but are specifically designed to practice the '389 Patent.

173. Indeed, as shown above, the Accused Products have no substantial non-infringing

uses because the accused functionality, including the behavioral "threat analysis" and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. For example, without compiling and analyzing data about the behavior of an object running on one or more remote computers, the Accused Products could not detect processes (running objects) that have made "[u]nusual changes to the registry" or to "search devices for signs of a suspected or known threat." (*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/learningContents/Live Discover.html.) These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the '389 Patent, that functionality could not be performed.

174. Additionally, the accused functionality, including the accused behavioral threat analysis functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without a "Threat Analysis Center" that compiles and correlates behavioral data from processes that run on the endpoints protected by the Accused Products, the Accused Products could not assess which behaviors on any given endpoint constitute a threat. These process are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the '389 Patent, that functionality could not be performed.

175. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Defendant constitute a material part of the invention—indeed, it provides all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and corresponding functionality (including the behavioral threat analysis functionality) constitute a material part of the inventions claimed because such analysis is integral to the processes identified above (such as to "correlating, by the base computer, at least a portion of the data about the computer object received from the first remote computer to at least a portion of the data about the computer object received from the second remote computer," "comparing, by the base computer, the correlated data about the computer object received from the first and second remote computers to other objects or entities to identify relationships between the correlated data and the other objects or entities," and "classifying, by the base computer, the computer object as malware on the basis of said comparison"), as required by the claims of the '389 Patent. None of these products are staple goods—they are sophisticated and customized cyber security and malware detection products and systems.

176. Defendant's infringing actions have continued after the original Complaint was filed. Defendant had knowledge of the '389 Patent, and of the specific conduct that constitutes infringement of the '389 Patent at least based on the original Complaint, but has continued to engage in the infringing activity, including by selling, making, configuring, and installing the Accused Products and performing the accused functionality, and by engaging in activities that constitute inducing infringement and contributory infringement as described above.

177. On information and belief, the infringing actions of each partner, customer, and/or end-user of the Accused Products are attributable to Defendant. For example, on information and

belief, Defendant directs and controls the activities or actions of its partners or others in connection with the Accused Products by contractual agreement or otherwise requiring partners or others to provide information and instructions to customers who acquire the Accused Products which, when followed, results in infringement. Defendant further directs and controls the operation of devices executing the Accused Products by programming the software which, when executed by a customer or end user, perform the method steps of at least claim 1 of the ’389 Patent.

178. Plaintiffs have suffered and continue to suffer damages, including lost profits, as a result of Defendant’s infringement of the ’389 Patent. Defendant is therefore liable to Plaintiffs under 35 U.S.C. § 284 for damages in an amount that adequately compensates Plaintiffs for Defendant’s infringement, but no less than a reasonable royalty.

179. Plaintiffs will continue to suffer irreparable harm unless this Court preliminarily and permanently enjoins Defendant, its agents, employees, representatives, and all others acting in concert with Defendant from infringing the ’389 Patent.

180. Defendant’s infringement of the ’389 Patent is knowing and willful. Defendant had actual knowledge of the ’389 Patent at least by the time Plaintiffs filed this lawsuit and had constructive knowledge of the ’389 Patent from at least the date Plaintiffs marked their products with the ’389 Patent and/or provided notice of the ’389 Patent on their website.

181. On information and belief, despite Defendant’s knowledge of the ’389 Patent, and Plaintiffs’ patented technology, Defendant made the deliberate decision to sell products and services that they knew infringe the ’389 Patent. Defendant’s continued infringement of the ’389 Patent with knowledge of the ’389 Patent constitutes willful infringement.

182. Plaintiffs’ allegations of infringement, indirect infringement, and willful infringement with respect to this patent are further set forth in Plaintiffs’ Disclosure of Asserted

Claims and Preliminary Infringement Contentions Pursuant to the Court's Standing Order Governing Patent Proceedings served July 12, 2022.

THIRD CAUSE OF ACTION
(INFRINGEMENT OF THE '045 PATENT)

183. Plaintiffs reallege and incorporates by reference the allegations of the preceding paragraphs of this Complaint.

184. Sophos has infringed and continues to infringe one or more claims of the '045 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including features such as Sophos' Intercept X Advanced with EDR ("Intercept X"), at least when used for their ordinary and customary purposes, practice each element of at least claim 1 of the '045 Patent as described below.

185. Claim 1 of the '045 Patent recites:

1. A method comprising:

gathering one or more events defining an action of a first object acting on a target;

generating a contextual state for at least one of the one or more events by correlating the at least one event to an originating object, the contextual state including an indication of the originating object of the first object and an indication of at least one of a device on which the first object is executed and a user associated with the first object;

obtaining a global perspective for the at least one event by obtaining information associated with one or more of the first object and the originating object, the information including at least one of age, popularity, a determination as to whether the first object is malware, a determination as to whether the originating object is malware, Internet Protocol (IP) Address, and Uniform Resource Locator (URL) information, wherein the global perspective for one or more related events to at least one event across a network;

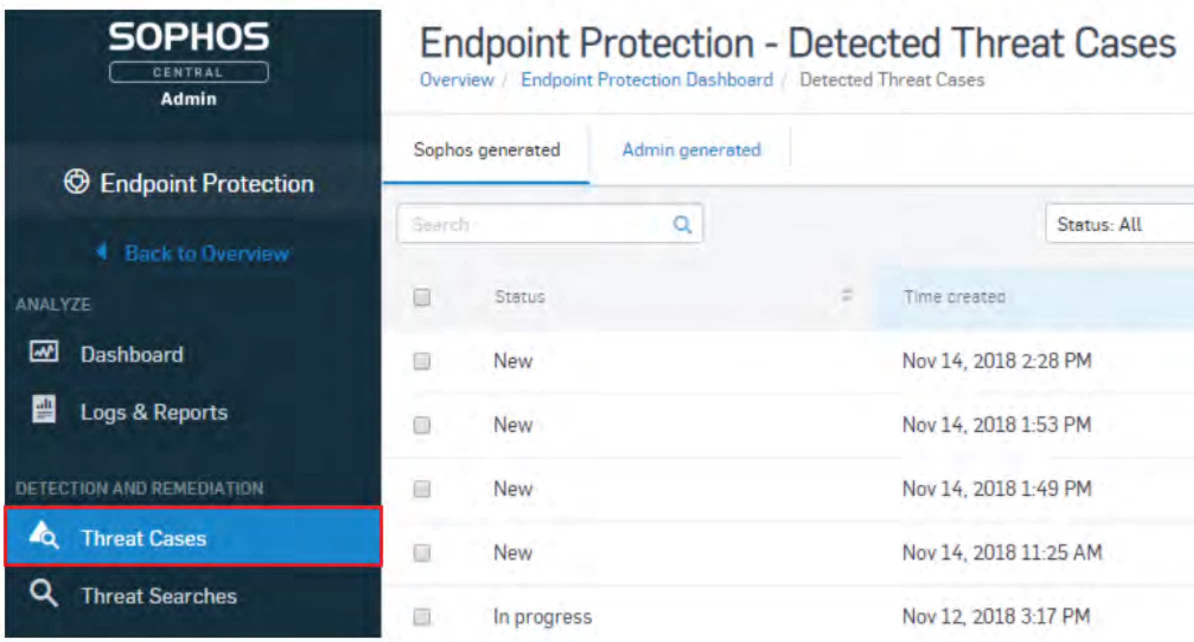
assembling an event line including details associated with the at least one event, the details including information uniquely identifying the first object, the action of the first object, the target, and the originating object; and

transmitting the assembled event line.

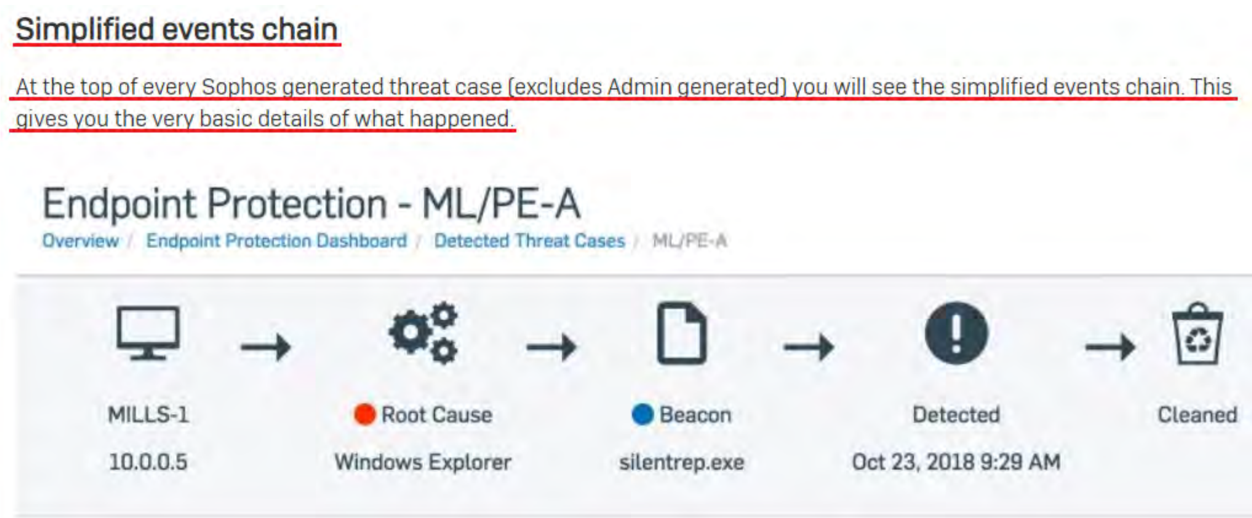
186. The Accused Products perform each of the method steps of claim 1 of the '045 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform a *method* as further explained below. For example, Intercept X performs a method for endpoint protection, wherein threat cases/attacks are analyzed in detail.

For customers with Sophos EDR, the full list of Threat Cases can be found in the below locations:

- Endpoint Protection > Detection and Remediation > Threat Cases
- Server Protection > Analyze > Threat Cases

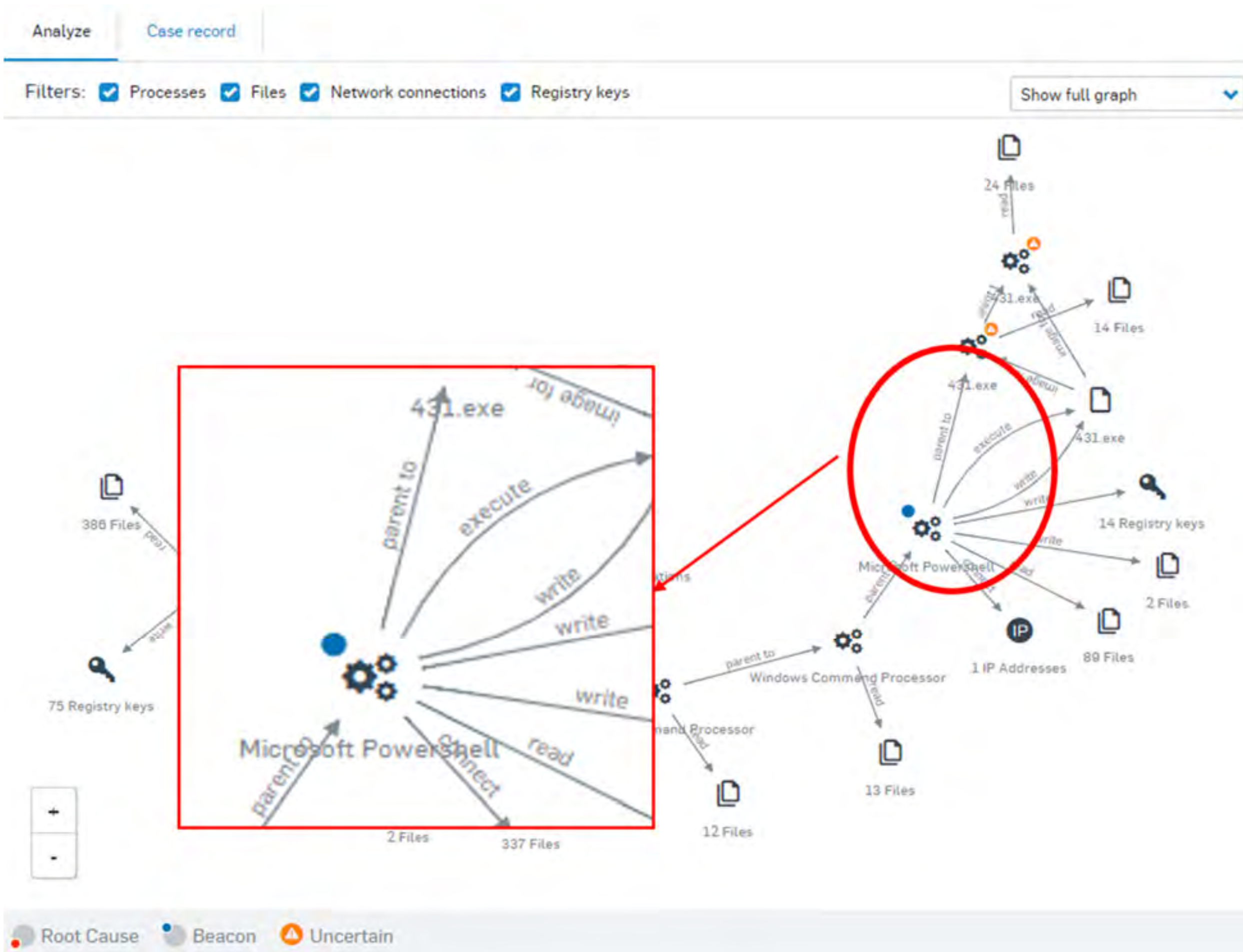


To view a Threat Case click on the detection **Name**:



(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl.)

187. The Accused Products perform a method that includes *gathering one or more events defining an action of a first object acting on a target*. In the example shown below, the "Analyze" tab of Intercept X illustrates the illicit creation of the malicious file "431.exe" by Microsoft Powershell, which is marked as the "Beacon," and the subsequent actions of "431.exe" on the infected endpoint, such as modifying registry keys. The Analyze Tab describes the "attack chain," *i.e.*, the chain of events linking the "Beacon" to what Intercept X has identified as the "Root Cause," in this case the program "Outlook." In the example below, Outlook wrote a document called "rgnr-avr11205-85.doc" and used Microsoft Office to read the document, initiating a chain of events culminating in the illicit creation and execution of the malicious file "431.exe" via Microsoft Powershell. Events defining the attack chain are gathered from the endpoint device by Intercept X (*e.g.*, Sophos Central).



(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl.)

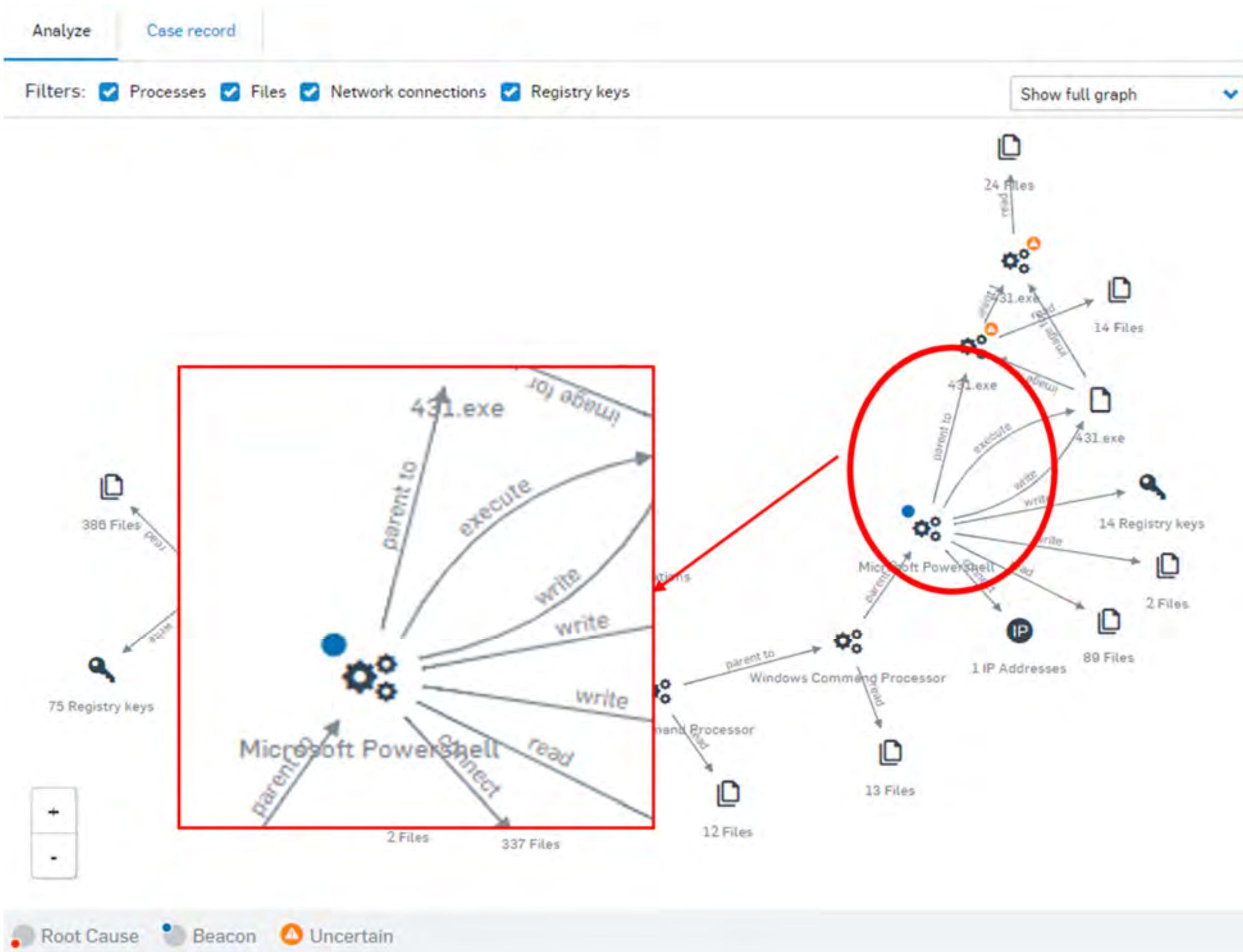
188. The root cause analysis performed by Intercept X, and illustrated by an attack chain, is further described in the video "Root Cause Analysis RCA in 2 minutes."

(*See* https://youtu.be/AOsjUjp4P7Q?t=48 (showing the root cause as the process circled in red above).)

189. The Accused Products perform a method that includes *generating a contextual state for at least one of the one or more events by correlating the at least one event to an originating object, the contextual state including an indication of the originating object of the first object and an indication of at least one of a device on which the first object is executed and a user associated with the first object*. As explained above, Intercept X's "Analyze" tab illustrates the creation of a malicious file "431.exe" by Microsoft Powershell, which is marked as the "Beacon."

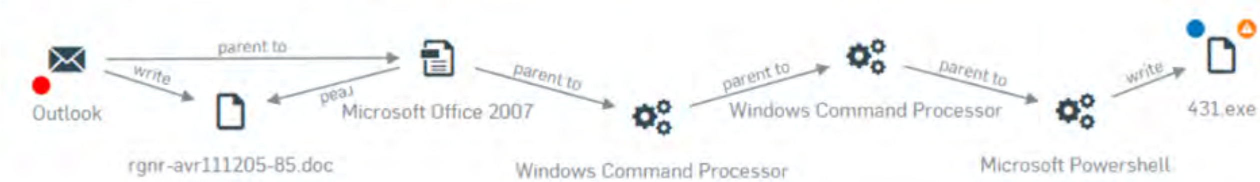
190. In the example below, the Analyze Tab describes the "attack chain," *i.e.*, the chain of associated events linking the "Beacon" to what Intercept X has identified as the "Root Cause," in this case the program "Outlook." Outlook wrote a document called "rgnr-avr11205-85.doc" and used Microsoft Office to read the document, initiating a chain of events culminating in the illicit creation and execution of the malicious file "431.exe" via Microsoft Powershell. An "attack chain," also known as an "event chain," is associated both with the endpoint device on which files

executed and the user of that device.



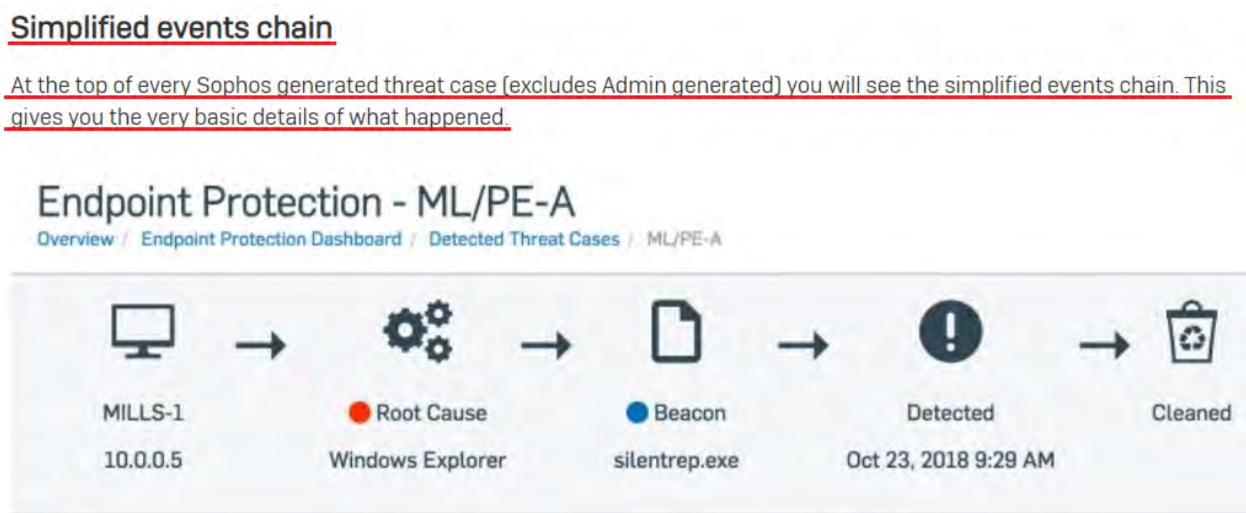
(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl.)

This hides most of the events leaving only the ones that directly link the root cause to the beacon. It is now easy to see that Outlook wrote a word document called rgnr-avr111205-85.doc, we can also see that Outlook launched a Microsoft Office application, which read the doc file.



(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-

us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html.)



(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language=en_USl.)

191. The details of the attack chain, and the relationships it illustrates between the "Root Cause," the "Beacon," and the intervening files or processes between them, include the "*contextual state for at least one of the one or more events by correlating the at least one event to an originating object, the contextual state including an indication of the originating object of the first object*." For example, the attack chain above shows each step in the attack (*e.g.*, files or registry keys read or written by any program, IP addresses accessed, caller-callee relationships, and more).

192. The Accused Products perform a method that includes *obtaining a global perspective for the at least one event by obtaining information associated with one or more of the first object and the originating object, the information including at least one of age, popularity, a determination as to whether the first object is malware, a determination as to whether the originating object is malware, Internet Protocol (IP) Address, and Uniform Resource Locator (URL) information, wherein the global perspective for one or more related events to the at least*

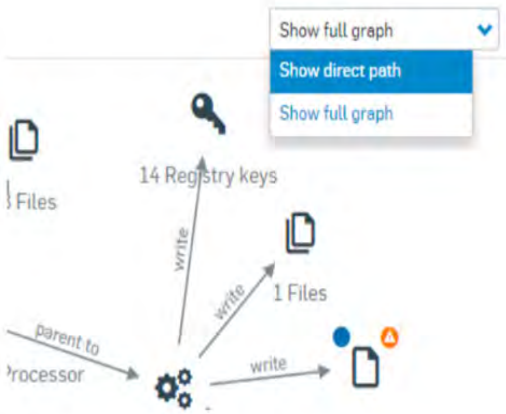
one event across a network. The attack chain includes information obtained about the "Root Cause," the "Beacon," events involving them, as well as the intervening files or processes from associated events across a network. For example, Intercept X obtains information at least about the age, popularity and URL information of the processes within the attack chain.

Report summary

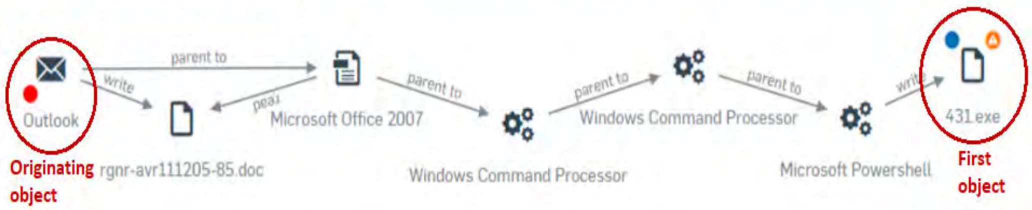
Under **Report summary**, you can see the file's reputation and prevalence and the results of our machine learning analysis, which indicate how suspicious the file is.

Setting	Description
Prevalence	Indicates how often SophosLabs has seen the file.
First seen	When SophosLabs first saw the file in the wild.
Last seen	When SophosLabs last saw the file in the wild
Machine learning analysis	Summarizes how suspicious the file is.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html.)



This hides most of the events leaving only the ones that directly link the root cause to the beacon. It is now easy to see that Outlook wrote a word document called rgnr-avr111205-85.doc, we can also see that Outlook launched a Microsoft Office application, which read the doc file.

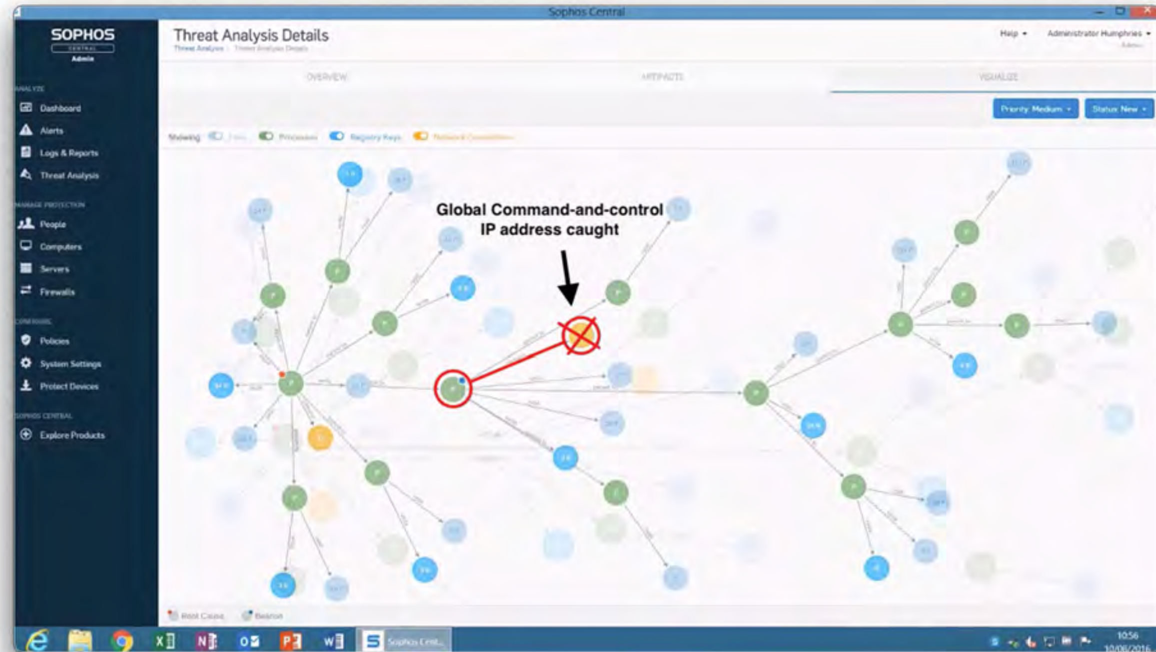


We can already see that the Microsoft Office event does not have a reputation icon, which means it has a good reputation.

Note: Reputation is only calculated for Portable Executable (PE) files, for example .exe, .dll. It is not shown for other file types such as .doc, .pdf, .png.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html*; see also* WBR_SOP000279, https://www.youtube.com/watch?v =_ujOT58ZvpI.)

193. Moreover, the Accused Products identify, for example, global IP addresses that are malware command-and-control centers: "Here's the beacon event, the thing that was caught, in this case a fake salary report. Here's the root, Google Chrome. We see here that the fake report reached out to an IP address that we happen to know to be a command-and-control site. This right here is the moment of conviction, when we discovered that what was executing was a piece of malware."

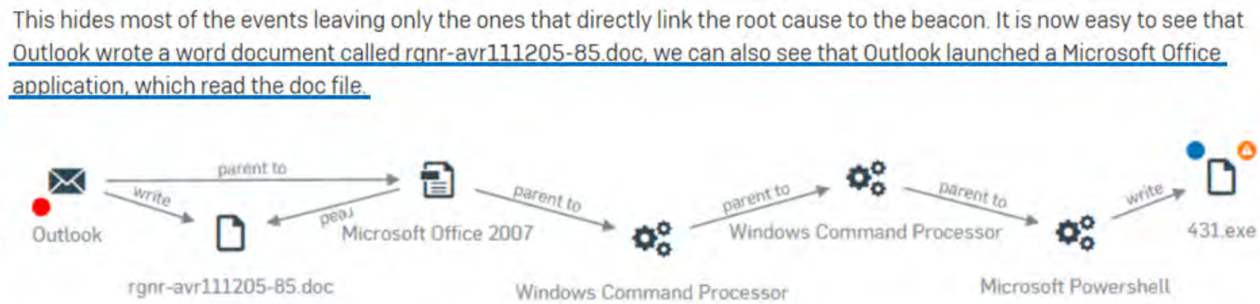


(*See* https://youtu.be/AOsjUjp4P7Q?t=48.)

194. The Accused Products perform a method that includes *assembling an event line including details associated with the at least one event, the details including information uniquely identifying the first object, the action of the first object, the target, and the originating object*. As explained above, an attack chain created by Intercept X includes information associated with, and identifying the "Root Cause," the "Beacon," and the intervening files or processes in the attack chain. In the example included, Intercept X's "Analyze" tab illustrates the illicit creation of the malicious file "431.exe" by Microsoft Powershell, and the subsequent actions of "431.exe" on the infected endpoint, such as modifying registry keys.

195. In the example below, the Analyze Tab describes the "attack chain," *i.e.*, the chain of events linking the "Beacon" to what Intercept X has identified as the "Root Cause," in this case the program "Outlook." Outlook wrote a document called "rgnr-avr11205-85.doc" and used Microsoft Office to read the document, initiating a chain of events culminating in the illicit creation

and execution of the malicious file “431.exe” via Microsoft Powershell. The attack chain, clearly seen as a chain of arrows connecting objects and their next targets, also illustrates the subsequent actions of “431.exe” on the infected endpoint, such as modifying registry keys. (*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl.)


This hides most of the events leaving only the ones that directly link the root cause to the beacon. It is now easy to see that Outlook wrote a word document called rgnr-avr111205-85.doc, we can also see that Outlook launched a Microsoft Office application, which read the doc file.

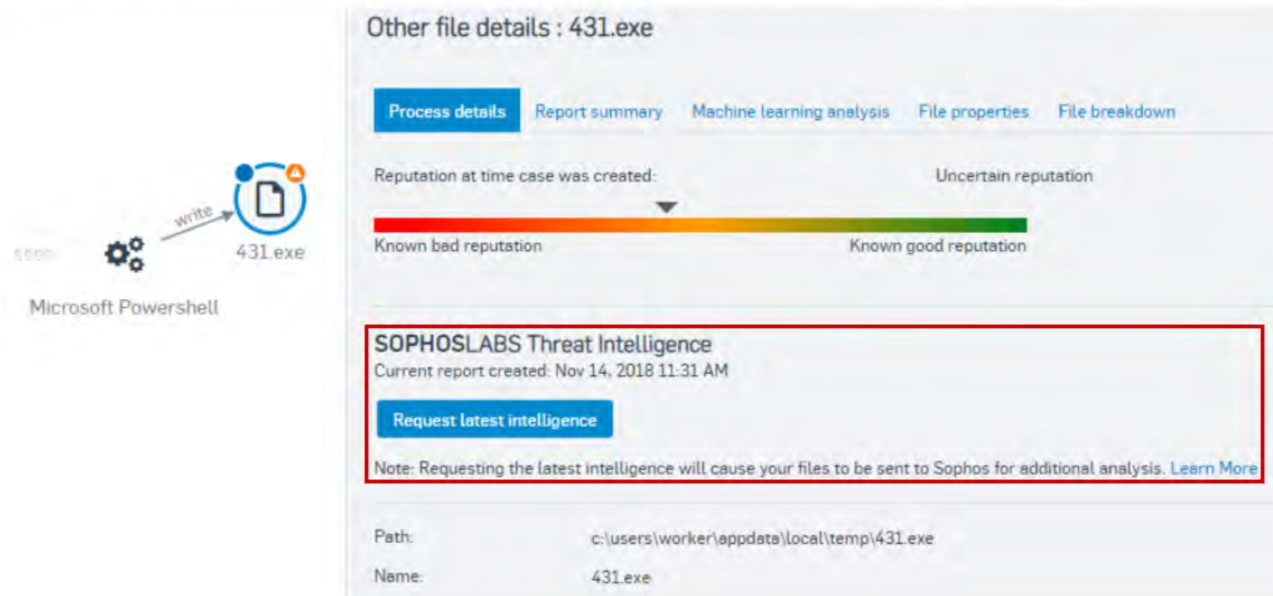
(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html.)

196. The Accused Products perform a method of *transmitting the assembled event line*. In the example below, and as explained above, Intercept X’s “Analyze” tab illustrates the “attack chain” linking the illicit creation of the malicious file “431.exe” by Microsoft Powershell, *i.e.*, “the Beacon,” to the “Root Cause.” Intercept X thus transmits the event line such that the attack chain can be generated, stored or displayed (*e.g.*, on a user or administrator’s client-side web browser).

197. Furthermore, Intercept X transmits the event line to SophosLABS for “additional analysis.”

Powershell has then written the 431.exe file which was detected by Sophos Deep Learning as ML/PE-A.

By selecting the beacon event we can see confirmation that it has an uncertain reputation, as well as that it was written to the users AppData location, this location is typically meant for data not executable's so this is also suspicious.



(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html.)

198. Each claim in the ’045 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ’045 Patent.

199. Defendant has been aware of the ’045 Patent since at least the filing of this Complaint. Further, Plaintiffs have marked its products with the ’045 Patent, including on its web site, since at least July 2020.

200. Defendant directly infringes at least claim 1 of the ’045 Patent, literally or under the doctrine of equivalents, by performing the steps described above. For example, on information and belief, Defendant performs the claimed method as described above by running this software and corresponding systems to protect its own computer and network operations. On information

and belief, Defendant also performs the claimed method as described above when testing the operation of the Accused Products and corresponding systems. As another example, Defendant performs the claimed method as described above when providing or administering services to third parties, customers, and partners using the Accused Products.

201. Defendant's partners, customers, and end users of the Accused Products and corresponding systems and services directly infringe at least claim 1 of the '045 Patent, literally or under the doctrine of equivalents, at least by using the Accused Products and corresponding systems and services, as described above.

202. Defendant has actively induced and is actively inducing infringement of at least claim 1 of the '045 Patent with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b). For example, Defendant encourages and induces customers with specific intent to use Sophos security software in a manner that infringes claim 1 of the '045 Patent at least by offering and providing software that performs a method that infringes claim 1 when installed and operated by the customer, and by engaging in activities relating to selling, marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including the activities described below.

203. Defendant encourages, instructs, directs, and/or requires third parties—including its certified partners and/or customers—to perform the claimed method using the software, services, and systems in infringing ways, as described above.

204. Defendant further encourages and induces its customers to infringe claim 1 of the '045 Patent: 1) by making its security services available on its website, providing applications that allow users to access those services, widely advertising those services, and providing technical support and instructions to users, and 2) through activities relating to marketing, advertising,

promotion, installation, support, and distribution of the Accused Products, including its Sophos security software, and services in the United States. (*See* https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx; WBR_SOP000525, https://partners.sophos.com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1; WBR_SOP000209, https://secure2.sophos.com/en-us/security-news-trends/whitepapers/gated-wp/cybersecurity-system-buyers-guide.aspx%23formFrame.)

205. For example, on information and belief, Defendant shares instructions, guides, and manuals, which advertise and instruct third parties on how to use the software as described above, including at least customers and partners. (*Id.*) On further information and belief, Sophos also provides customer service and technical support to purchasers of the Accused Products and corresponding system and services, which directs and encourages customers to perform certain actions that use the Accused Products in an infringing manner. (*Id.*)

206. Defendant and/or its partners recommend and sell the Accused Products and provide technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Defendant and/or one of its partners, which obligates each customer to perform certain actions as a condition to use of the Accused Products. Further, in order to receive the benefit of Defendant and/or its partner's continued technical support and their specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that infringes the '045 Patent.

207. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Product remains operational for each customer through ongoing technical support, on information and belief, Defendant and/or its

partners affirmatively aid and abet each customer's use of the Accused Products in a manner that performs the claimed method of, and infringes, the '045 Patent.

208. Defendant also contributes to the infringement of their partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the claimed methods, and not a staple article or commodity of commerce suitable for substantial non-infringing uses. Indeed, as shown above, the Accused Products and the example functionality described below have no substantial non-infringing uses but are specifically designed to practice the '045 Patent.

209. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the "Root Cause Analysis," "Threat Analysis," and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. For example, without analyzing data about the behavior and globally known information about the maliciousness of an object running on one or more remote computers, the Accused Products could not detect when that object had engaged in an illicit behavior, and could not create a forensic trail associating that object with its root cause. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice in the '045 Patent, that functionality could not be performed.

210. Additionally, the accused functionality, including the "Root Cause Analysis,"

"Threat Analysis," and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without creating an "attack chain" that compiles and correlates behavioral data from processes that run on the endpoints protected by the Accused Products, the Accused Products could not describe how and why the illicit behaviors detected on any given endpoint constitute a threat. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the '045 Patent, that functionality could not be performed.

211. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Defendant constitute a material part of the invention--indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionalities (including the "Root Cause Analysis"/"Threat Analysis" functionality) constitute a material part of the inventions claimed at least because they are integral to the processes identified above (such as "generating a contextual state…"; "obtaining a global perspective…"; "assembling" and "transmitting" an event line"), as recited in the claims of the '045 Patent. None of these products are staple goods—they are sophisticated and customized cyber security and malware detection products, methods, and systems.

212. Defendant's infringing actions have continued after the original Complaint was filed. Defendant had knowledge of the '045 Patent, and of the specific conduct that constitutes

infringement of the ’045 Patent at least based on the original Complaint, but has continued to engage in the infringing activity, including by selling making, configuring, and installing the Accused Products and performing the accused functionality, and by engaging in activities that constitute inducing infringement and contributory infringement as described above.

213. On information and belief, the infringing actions of each partner, customer, and/or end-user of the Accused Products are attributable to Defendant. For example, on information and belief, Defendant directs and controls the activities or actions of its partners or others in connection with the Accused Products by contractual agreement or otherwise requiring partners or others to provide information and instructions to customers who acquire the Accused Products which, when followed, results in infringement. Defendant further directs and controls the operation of devices executing the Accused Products by programming the software which, when executed by a customer or end user, perform the method steps of at least claim 1 of the ’045 Patent.

214. Plaintiffs have suffered and continue to suffer damages, including lost profits, as a result of Defendant’s infringement of the ’045 Patent. Defendant is therefore liable to Plaintiffs under 35 U.S.C. § 284 for damages in an amount that adequately compensates Plaintiffs for Defendant’s infringement, but no less than a reasonable royalty.

215. Plaintiffs will continue to suffer irreparable harm unless this Court preliminarily and permanently enjoins Defendant, its agents, employees, representatives, and all others acting in concert with Defendant from infringing the ’045 Patent.

216. Defendant’s infringement of the ’045 Patent is knowing and willful. Defendant had actual knowledge of the ’045 Patent at least by the time Plaintiffs filed this lawsuit and had constructive knowledge of the ’045 Patent from at least the date Plaintiffs marked their products with the ’045 Patent and/or provided notice of the ’045 Patent on their website.

217. On information and belief, despite Defendant's knowledge of the Asserted Patents and Plaintiffs' patented technology, Defendant made the deliberate decision to sell products and services that they knew infringe the '045 Patent. Defendant's continued infringement of the '045 Patent with knowledge of the '045 Patent constitutes willful infringement.

218. Plaintiffs' allegations of infringement, indirect infringement, and willful infringement with respect to this patent are further set forth in Plaintiffs' Disclosure of Asserted Claims and Preliminary Infringement Contentions Pursuant to the Court's Standing Order Governing Patent Proceedings served July 12, 2022.

FOURTH CAUSE OF ACTION
(INFRINGEMENT OF THE '224 PATENT)

219. Plaintiffs reallege and incorporate by reference the allegations of the preceding paragraphs of this Complaint.

220. Sophos has infringed and continues to infringe one or more claims of the '224 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including features such as Sophos's Intercept X Advanced with EDR ("Intercept X"), at least when used for their ordinary and customary purposes, practice each element of at least claim 1 of the '224 Patent as described below.

221. Claim 1 of the '224 Patent recites:

1. A method comprising:

gathering an event defining an action of a first object acting on a target, wherein the first object is executed on a device;

generating contextual state information for the event by correlating the event to an originating object of the first object;

obtaining a global perspective for the event based on the contextual state

information, wherein the global perspective comprises information associated with one or more of the first object and the originating object, and wherein the global perspective relates to one or more other events related to the event across a network;

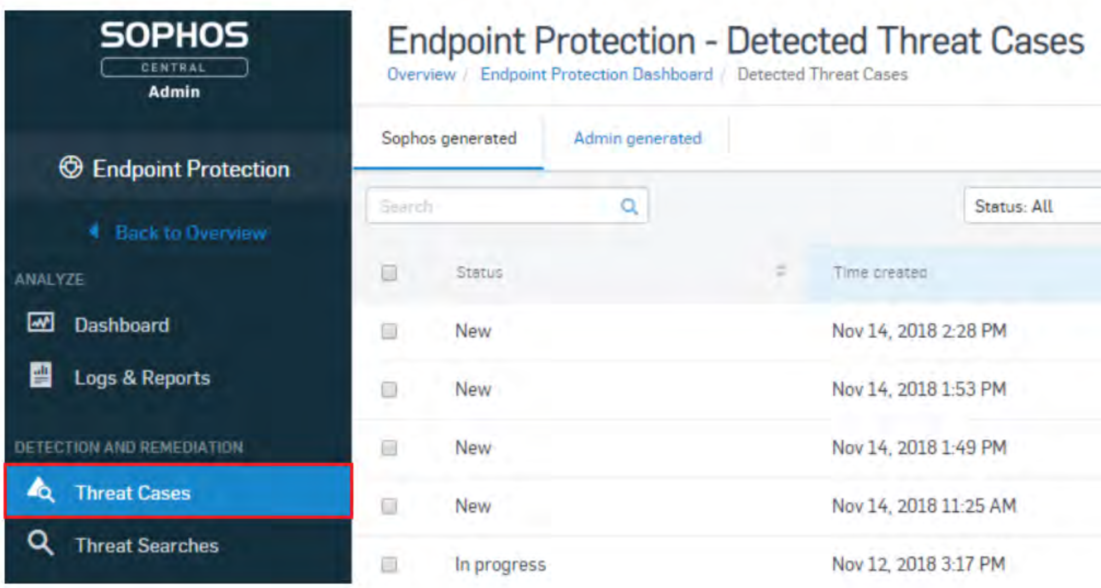
generating an event line comprising information relating to the event, wherein the information relates to at least one of the first object, the action of the first object, the target, and the originating object; and

transmitting the generated event line.

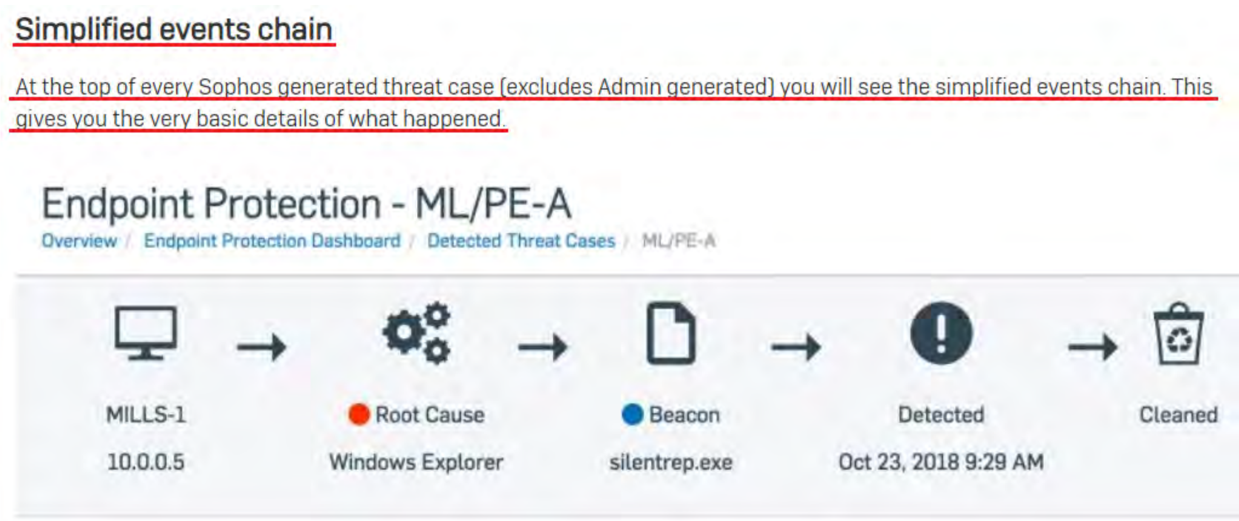
222. To the extent the preamble is construed to be limiting, the Accused Products perform a *method* as further explained below. For example, the Accused Products perform a method for endpoint protection, wherein threat cases/attacks are analyzed in detail.

For customers with Sophos EDR, the full list of Threat Cases can be found in the below locations:

- Endpoint Protection > Detection and Remediation > Threat Cases
- Server Protection > Analyze > Threat Cases



To view a Threat Case click on the detection **Name**:



(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language=en_USl.)

223. The Accused Products perform a method of *gathering an event defining an action of a first object acting on a target*, *wherein the first object is executed on a device*. In the example shown below, the "Analyze" tab of Intercept X illustrates the illicit creation of the malicious file "431.exe" by Microsoft Powershell, which is marked as the "Beacon," and the subsequent actions of "431.exe" on the infected endpoint, such as modifying registry keys. The Analyze Tab describes the "attack chain," *i.e.*, the chain of events linking the "Beacon" to what Intercept X has identified as the "Root Cause," in this case the program "Outlook." In the example below, Outlook wrote a document called "rgnr-avr11205-85.doc" and used Microsoft Office to read the document, initiating a chain of events culminating in the illicit creation and execution of the malicious file "431.exe" via Microsoft Powershell. Events defining the attack chain are gathered from the endpoint device by Intercept X (*e.g.*, Sophos Central). (*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language=en_US.)

224. The root cause analysis performed by Intercept X, and illustrated by an attack chain, is further described in the video "Root Cause Analysis RCA in 2 minutes":

(*See* https://youtu.be/AOsjUjp4P7Q?t=48 (showing the root cause as the process circled in red above).)

225. The Accused Products perform a method of *generating contextual state information for the event by correlating the event to an originating object of the first object*. As explained above, Intercept X's "Analyze" tab illustrates the illicit creation and execution of the malicious file "431.exe" by Microsoft Powershell, which is marked as the "Beacon."

226. In the example below, the Analyze Tab describes the "attack chain," *i.e.*, the chain of associated events linking the "Beacon" to what Intercept X has identified as the "Root Cause," in this case the program "Outlook." Outlook wrote a document called "rgnr-avr11205-85.doc" and used Microsoft Office to read the document, initiating a chain of events culminating in the illicit creation and execution of the malicious file "431.exe" via Microsoft Powershell. The attack chain also illustrates the subsequent actions of "431.exe" on the infected endpoint, such as modifying registry keys. (*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_US.)

This hides most of the events leaving only the ones that directly link the root cause to the beacon. It is now easy to see that Outlook wrote a word document called rgnr-avr111205-85.doc, we can also see that Outlook launched a Microsoft Office application, which read the doc file.

(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl.)

227. The details of the attack chain, and the relationships it illustrates between the "Root Cause," the "Beacon," and the intervening files or processes between them, include the "*contextual state information for the event by correlating the event to an originating object of the first object*."

228. The Accused Products perform a method of *obtaining a global perspective for the event based on the contextual state information wherein the global perspective comprises information associated with one or more of the first object and the originating object, and wherein the global perspective relates to one or more other events related to the event across a network.* The attack chain includes information obtained about the "Root Cause," the "Beacon," events involving them, as well as the intervening files or processes from associated events across a network. For example, Intercept X obtains information at least about the age, popularity URL information of the processes within the attack chain.

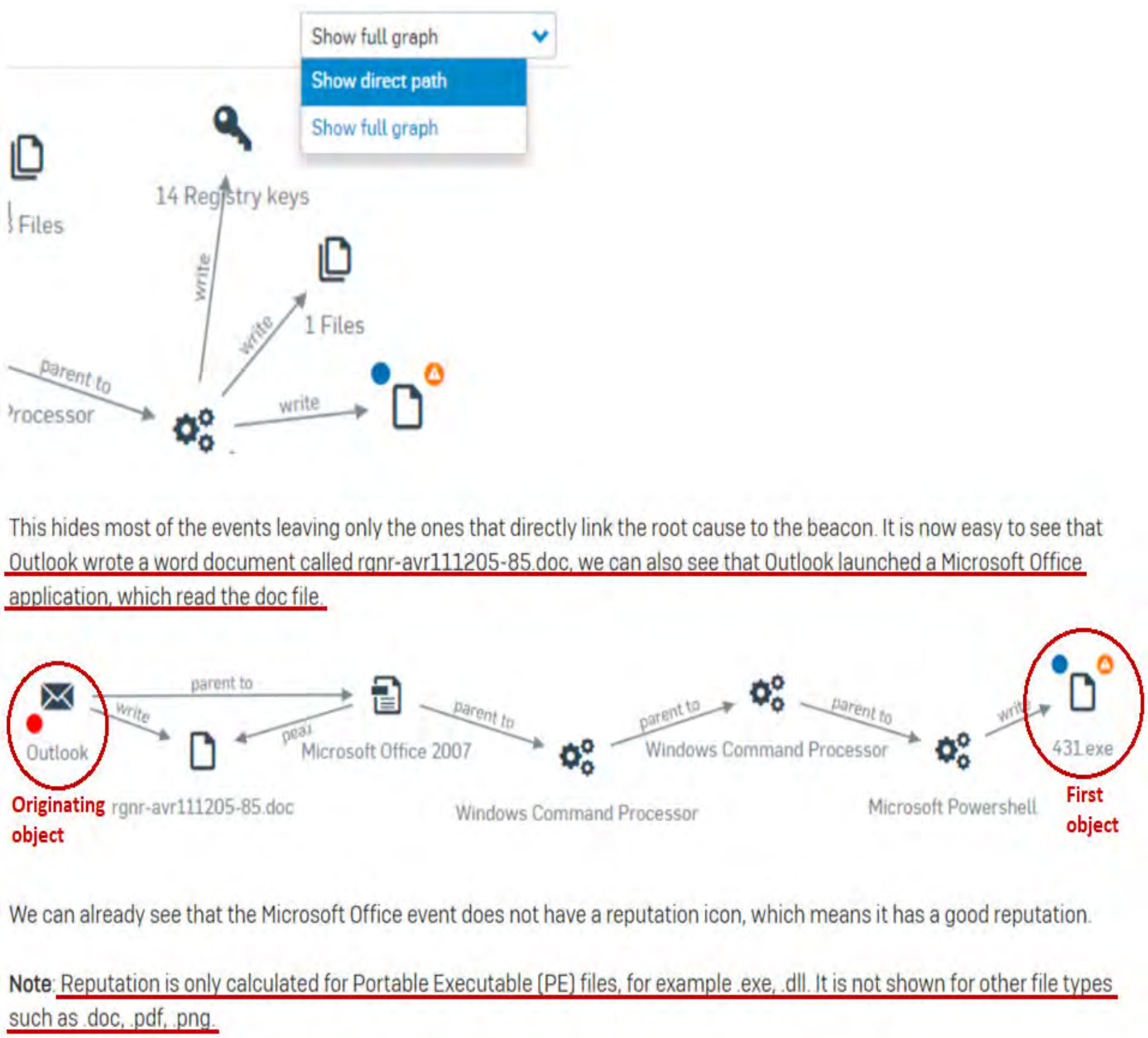
Report summary

Under **Report summary**, you can see the file's reputation and prevalence and the results of our machine learning analysis, which indicate how suspicious the file is.

Setting	Description
Prevalence	Indicates how often SophosLabs has seen the file.
First seen	When SophosLabs first saw the file in the wild.
Last seen	When SophosLabs last saw the file in the wild

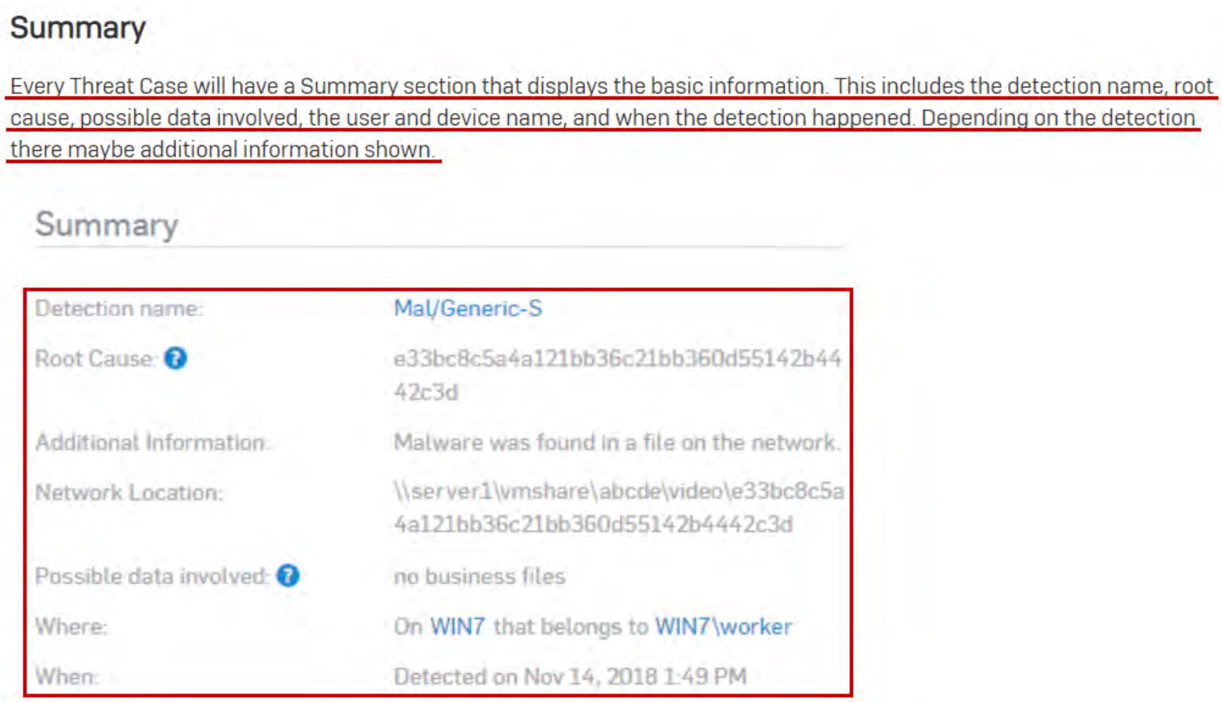
Setting	Description
Machine learning analysis	Summarizes how suspicious the file is.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html.)



(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-

us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html.)



Summary

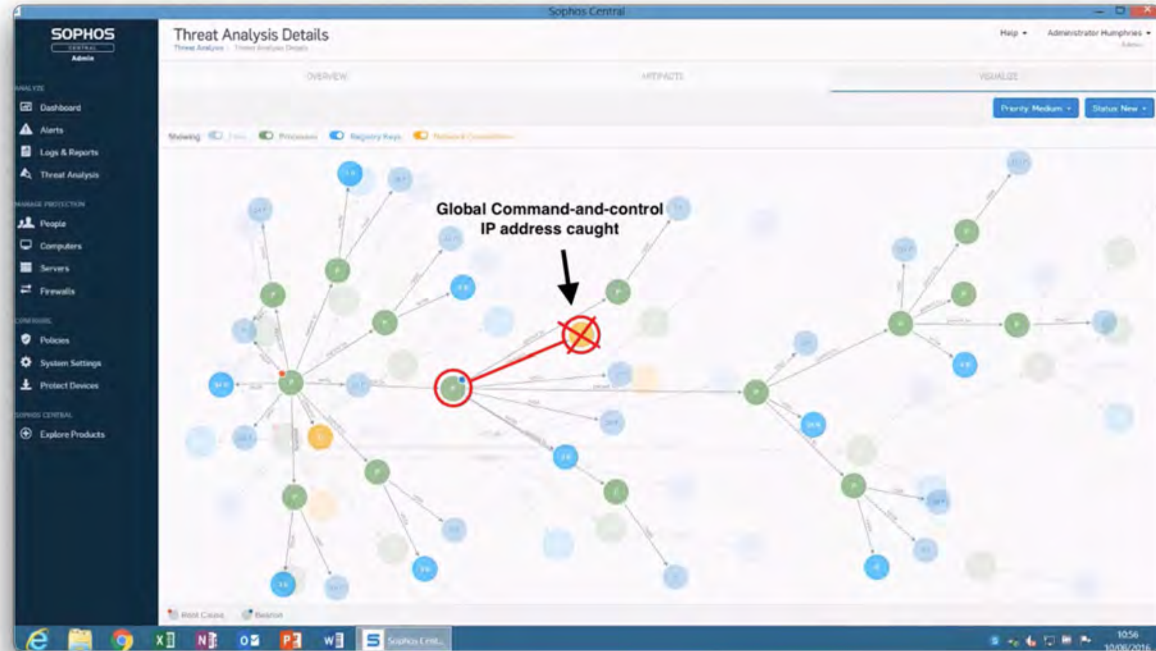
Every Threat Case will have a Summary section that displays the basic information. This includes the detection name, root cause, possible data involved, the user and device name, and when the detection happened. Depending on the detection there maybe additional information shown.

Summary

Detection name:	Mal/Generic-S
Root Cause:	e33bc8c5a4a121bb36c21bb360d55142b4442c3d
Additional Information:	Malware was found in a file on the network.
Network Location:	\\server1\vmshare\abcde\video\e33bc8c5a4a121bb36c21bb360d55142b4442c3d
Possible data involved:	no business files
Where:	On WIN7 that belongs to WIN7\worker
When:	Detected on Nov 14, 2018 1:49 PM

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html; *see also* WBR_SOP000279, https://www.youtube.com/watch?v=_ujOT58ZvpI.)

229. Moreover, the Accused Products identify, for example, global IP addresses that are malware command-and-control centers: "Here's the beacon event, the thing that was caught, in this case a fake salary report. Here's the root, Google Chrome. We see here that the fake report reached out to an IP address that we happen to know to be a command-and-control site. This right here is the moment of conviction, when we discovered that what was executing was a piece of malware."

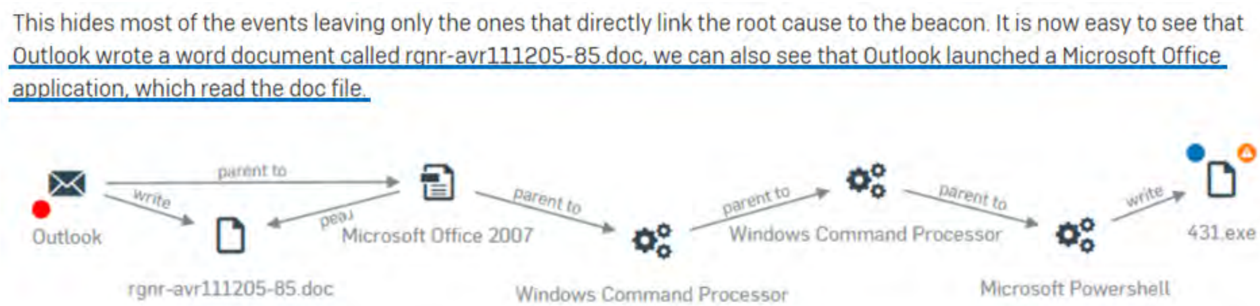


(*See* https://youtu.be/AOsjUjp4P7Q?t=48.)

230. The Accused Products perform a method of *generating an event line comprising information relating to the event, wherein the information relates to at least one of the first object, the action of the first object, the target, and the originating object*. As explained above, an attack chain created by Intercept X includes information associated with, and identifying the "Root Cause," the "Beacon," and the intervening files or processes in the attack chain. In the example included, Intercept X generates an attack chain illustrating the illicit creation, execution, and subsequent actions of the malicious file "431.exe."

231. In the example below, the "attack chain" links the "Beacon" to what Intercept X has identified as the "Root Cause," in this case the program "Outlook." Outlook wrote a document called "rgnr-avr11205-85.doc" and used Microsoft Office to read the document, initiating a chain of events culminating in the illicit creation and execution of the malicious file "431.exe" via Microsoft Powershell. The attack chain also illustrates the subsequent actions of "431.exe" on the

infected endpoint, such as modifying registry keys. (*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language=en_US.)



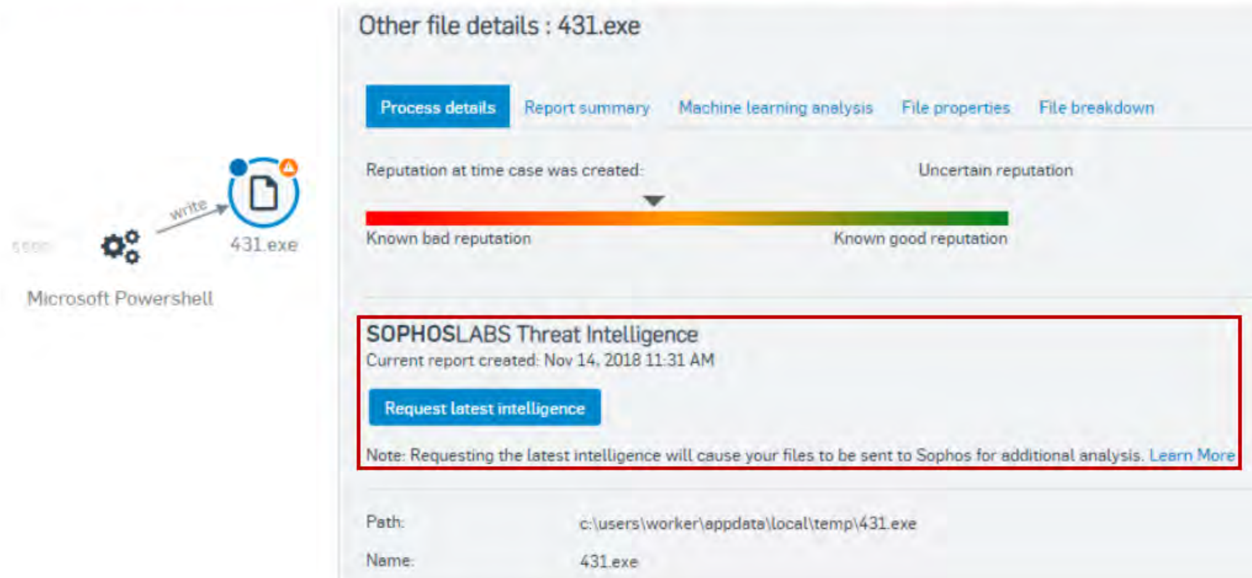
(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html.)

232. The Accused Products perform a method of *transmitting the generated event line*. In the example below, and as explained above, Intercept X's "Analyze" tab illustrates the "attack chain" linking the illicit creation, execution, and subsequent actions of the malicious file "431.exe" by Microsoft Powershell, *i.e.*, "the Beacon," to the "Root Cause." Intercept X thus transmits the event line such that the attack chain can be generated, stored or displayed.

233. Furthermore, Intercept X transmits the event line to SophosLABS for "additional analysis."

Powershell has then written the 431.exe file which was detected by Sophos Deep Learning as ML/PE-A.

By selecting the beacon event we can see confirmation that it has an uncertain reputation, as well as that it was written to the users AppData location, this location is typically meant for data not executable's so this is also suspicious.



(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html.)

For customers using Sophos EDR, by pressing the **Request latest intelligence** button, the file will be retrieved out of the Sophos quarantine and submitted to SophosLabs. A couple of minutes later the four other tabs (Report summary, Machine learning analysis, File properties, File breakdown) pictured will be displayed. The purpose of these these additional tabs is to help display the various properties of the file in a simple way. This can be useful for various reasons, one of them is to feel confident that the file is indeed malicious and not something you want in your environment. For more information on SophosLabs Threat Intelligence, please see: Sophos Central: Threat intelligence overview.

Now that we understand that the file was malicious and that it came from an email, which then used Microsoft Word, CMD and Powershell to execute the attack chain, we can decide what could be done to help prevent this type of attack happening again.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html.)

234. Each claim in the ’224 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ’224 Patent.

235. Defendant has been aware of the ’224 Patent since at least the filing of this Complaint. Further, Plaintiffs have marked its products with the ’224 Patent, including on its web site, since at least July 2020.

236. Defendant directly infringes at least claim 1 of the ’224 Patent, literally or under the doctrine of equivalents, by performing the steps described above. For example, on information and belief, Defendant performs the claimed method as described above by running this software and corresponding systems to protect its own computer and network operations. On information and belief, Defendant also performs the claimed method as described above when testing the operation of the Accused Products and corresponding systems. As another example, Defendant performs the claimed method as described above when providing or administering services to third parties, customers, and partners using the Accused Products.

237. Defendant’s partners, customers, and end users of the Accused Products and corresponding systems and services directly infringe at least claim 1 of the ’224 Patent, literally or under the doctrine of equivalents, at least by using the Accused Products and corresponding systems and services, as described above.

238. Defendant has actively induced and is actively inducing infringement of at least claim 1 of the ’224 Patent with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b). For example, Defendant encourages and induces customers with specific intent to use Sophos’ security software in a manner that infringes claim 1 of the ’224 Patent at least by offering and providing software that performs a method that infringes claim 1 when installed and operated by the customer, and by

engaging in activities relating to selling, marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including the activities described below.

239. Defendant encourages, instructs, directs, and/or requires third parties—including its certified partners and/or customers—to perform the claimed method using the software, services, and systems in infringing ways, as described above.

240. Defendant further encourages and induces its customers to infringe claim 1 of the '224 Patent: 1) by making its security services available on its website, providing applications that allow users to access those services, widely advertising those services, and providing technical support and instructions to users, and 2) through activities relating to marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including its Sophos security software, and services in the United States. (*See* https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx; WBR_SOP000525, https://partners.sophos.com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1; WBR_SOP000209, https://secure2.sophos.com/en-us/security-news-trends/whitepapers/gated-wp/cybersecurity-system-buyers-guide.aspx%23formFrame.)

241. For example, on information and belief, Defendant shares instructions, guides, and manuals, which advertise and instruct third parties on how to use the software as described above, including at least customers and partners. (*Id*.) On further information and belief, Defendant also provides customer service and technical support to purchasers of the Accused Products and corresponding system and services, which directs and encourages customers to perform certain actions that use the Accused Products in an infringing manner. (*Id*.)

242. Defendant and/or its partners recommend and sell the Accused Products and provide technical support for the installation, implementation, integration, and ongoing operation

of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Defendant and/or one of its partners, which obligates each customer to perform certain actions in order to use of the Accused Products. Further, in order to receive the benefit of Defendant and/or its partner's continued technical support and their specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that performs the claimed method and infringes the '224 Patent.

243. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Product remains operational for each customer through ongoing technical support, on information and belief, Defendant and/or its partners affirmatively aid and abet each customer's use of the Accused Products in a manner that performs the claimed method of, and infringes, the '224 Patent.

244. Defendant also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the claimed methods, and not a staple article or commodity of commerce suitable for substantial non-infringing uses. Indeed, as shown above, the Accused Products and the example functionality described below have no substantial non-infringing uses but are specifically designed to practice the '224 Patent.

245. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the "Threat Analysis," "Root Cause Analysis," and related functionality described above, is an integral part of the Accused Products and must be

performed for the Accused Products to perform their intended purpose. For example, without analyzing data about the behavior and globally known information about the maliciousness of an object running on one or more remote computers, the Accused Products could not detect when that object had engaged in an illicit behavior, and could not create a forensic trail associating that object with its root cause. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’224 Patent, that functionality could not be performed.

246. Additionally, the accused functionality, including the “Threat Analysis,” “Root Cause Analysis,” and related functionality, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without creating an “attack chain” that compiles and correlates behavioral data from processes that run on the endpoints protected by the Accused Products, the Accused Products could not describe how and why the illicit behaviors detected on any given endpoint constitute a threat. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’224 Patent, that functionality could not be performed.

247. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Defendant constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and

systems. For example, the Accused Products and accused functionalities (including the "Threat Analysis"/"Root Cause Analysis") constitute a material part of the inventions claimed at least because they are integral to the processes identified above (such as "generating contextual state information…"; "obtaining a global perspective…"; "generating" and "transmitting" an "event line"), as recited in the claims of the '224 Patent. None of these products are staple goods—they are sophisticated and customized cyber security and malware detection products, methods, and systems.

248. Defendant's infringing actions have continued after the original Complaint was filed. Defendant had knowledge of the '224 Patent, and of the specific conduct that constitutes infringement of the '224 Patent at least based on the original Complaint, but has continued to engage in the infringing activity, including by selling, making, configuring, and installing the Accused Products and performing the accused functionality, and by engaging in activities that constitute inducing infringement and contributory infringement as described above.

249. On information and belief, the infringing actions of each partner, customer, and/or end-user of the Accused Products are attributable to Defendant. For example, on information and belief, Defendant directs and controls the activities or actions of its partners or others in connection with the Accused Products by contractual agreement or otherwise requiring partners or others to provide information and instructions to customers who acquire the Accused Products which, when followed, results in infringement. Defendant further directs and controls the operation of devices executing the Accused Products by programming the software which, when executed by a customer or end user, perform the method steps of at least claim 1 of the '224 Patent.

250. Plaintiffs have suffered and continue to suffer damages, including lost profits, as a result of Defendant's infringement of the '224 Patent. Defendant is therefore liable to Plaintiffs

under 35 U.S.C. § 284 for damages in an amount that adequately compensates Plaintiffs for Defendant's infringement, but no less than a reasonable royalty.

251. Plaintiffs will continue to suffer irreparable harm unless this Court preliminarily and permanently enjoins Defendant, its agents, employees, representatives, and all others acting in concert with Defendant from infringing the '224 Patent.

252. Defendant's infringement of the '224 Patent is knowing and willful. Defendant had actual knowledge of the '224 Patent at least by the time Plaintiffs filed this lawsuit and had constructive knowledge of the '224 Patent from at least the date Plaintiffs marked their products with the '224 Patent and/or provided notice of the '224 Patent on their website.

253. On information and belief, despite Defendant's knowledge of the Asserted Patents and Plaintiffs' patented technology, Defendant made the deliberate decision to sell products and services that they knew infringe the '224 Patent. Defendant's continued infringement of the '224 Patent with knowledge of the '224 Patent constitutes willful infringement.

254. Plaintiffs' allegations of infringement, indirect infringement, and willful infringement with respect to this patent are further set forth in Plaintiffs' Disclosure of Asserted Claims and Preliminary Infringement Contentions Pursuant to the Court's Standing Order Governing Patent Proceedings served July 12, 2022.

FIFTH CAUSE OF ACTION
(INFRINGEMENT OF THE '591 PATENT)

255. Plaintiffs reallege and incorporate the preceding paragraphs of this complaint.

256. Defendant has infringed and continues to infringe one or more claims of the '591 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including anti-exploit features such as those included in Intercept X, at least when used for their ordinary and customary

purposes, practice each element of at least claim 1 of the ’591 Patent as described below.

257. For example, claim 1 of the ’591 Patent recites:

> 1. A computer-implemented method comprising:
>
> monitoring a memory space of a process for execution of at least one monitored function of a plurality of functions, wherein monitoring the memory space comprises loading a component for evaluating the at least one monitored function in the memory space;
>
> invoking one of the plurality of functions as a result of receiving a call from an application programming instance;
>
> executing stack walk processing upon the invocation of one of the plurality of functions in the monitored memory space; and
>
> performing, during the executing of the stack walk processing before an address of an originating caller function is reached, a memory check for a plurality of stack entries identified during the stack walk processing to detect suspicious behavior, wherein an alert of suspicious behavior is triggered when the performing of the memory check detects at least one of the following:
>
> code execution is attempted from non-executable memory,
>
> a base pointer is identified as being invalid,
>
> an invalid stack return address is identified,
>
> attempted execution of a return-oriented programming technique is detected,
>
> the base pointer is detected as being outside a current thread stack, and
>
> a return address is detected as being inside a virtual memory area,
>
> wherein when an alert of suspicious behavior is triggered, preventing execution of a payload for the invoked function from operating.

258. The Accused Products perform each of the method steps of claim 1 of the ’591 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform *a computer-implemented method*, as further explained below. For example, the Accused Products include “[e]xploit prevention [to] stop[] the techniques used in file-less, malware-less, and exploit-

based attacks." (*See* WBR_SOP000222, https://www.sophos.com/en-us/products/endpoint-antivirus.aspx.)

259. The Accused Products perform a method that includes *monitoring a memory space of a process for execution of at least one monitored function of a plurality of functions, wherein monitoring the memory space comprises loading a component for evaluating the at least one monitored function in the memory space*. For example, the Accused Products load a component for monitoring memory space when monitoring "sensitive API functions." In another example, the Accused Products include "Memory Scanning" for "defense against in-memory malware" and monitor "API call[s] (*e.g.*, VitrualAlloc)."

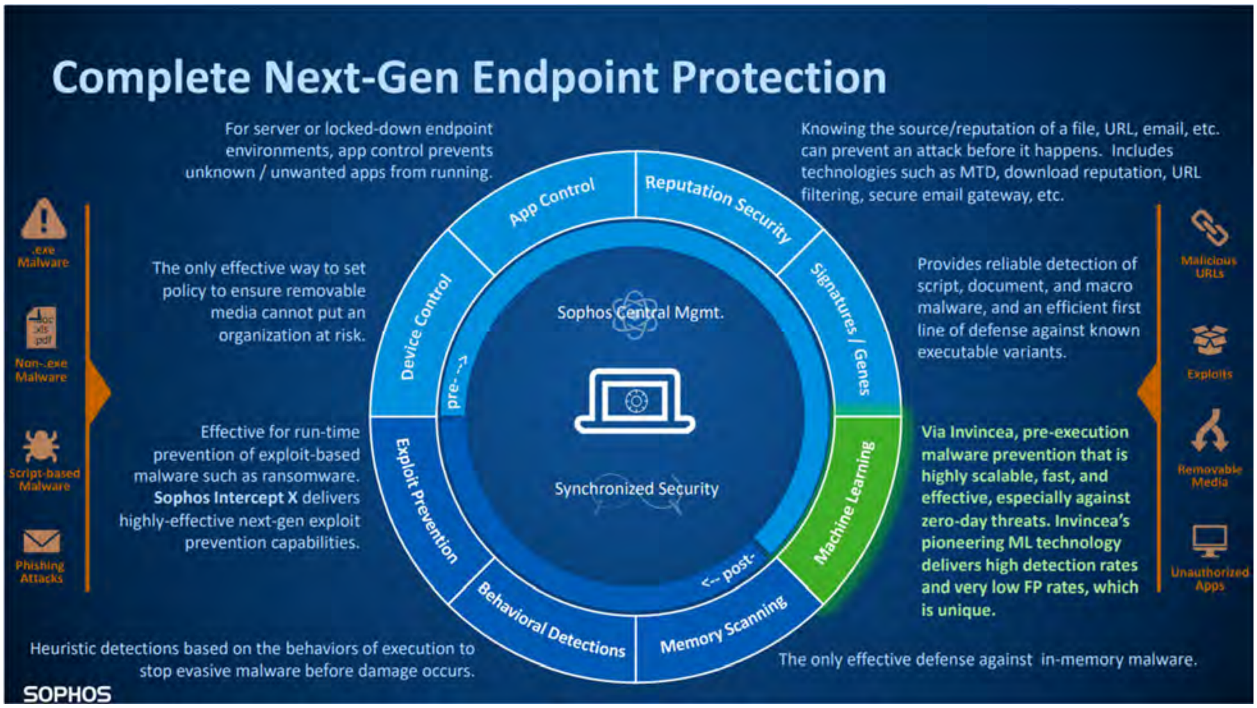
Stack-based ROP Mitigation (Caller)

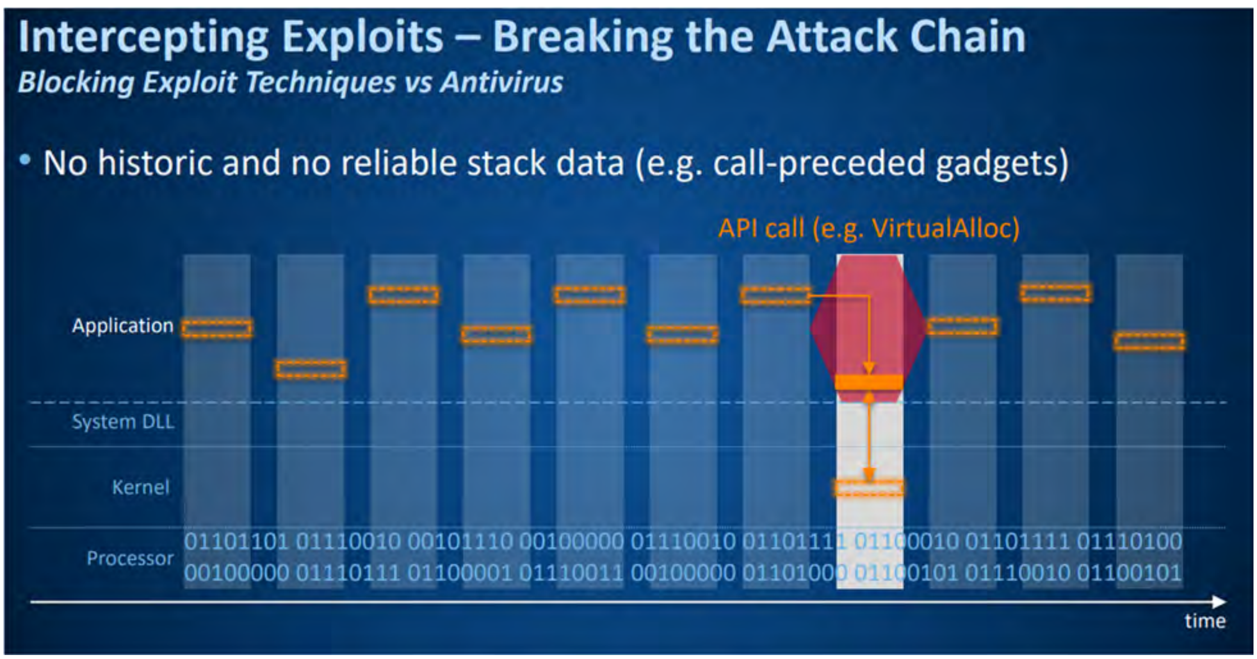
To defeat security technologies like data execution prevention (DEP) and address space layout randomization (ASLR), attackers typically resort to hijacking control-flow of vulnerable internet-facing applications. Such in-memory attacks are invisible to antivirus, most "next-gen" products, and other cyber defenses as there are no malicious files involved. Instead, the attack is constructed at run time by combining short pieces of benign code that are part of existing applications like Internet Explorer and Adobe Flash Player – a so-called code-reuse or return-oriented programming (ROP) attack.

During normal control-flow, sensitive API functions – like VirtualAlloc and CreateProcess – are invoked by the CALL instruction. Upon invoking a sensitive API, typical ROP defenses stop code execution to determine the API invoking address, using the 'return' address which is located on top of the stack. If the instruction of the API invoking address is not a CALL, the process is terminated.

Since the contents of the stack are writable, an attacker can write specific values on the stack to mislead the analysis of the stack-based ROP defense. The stack-based ROP defense cannot determine if the contents of the stack are benign or manipulated by an attacker.

(*See* WBR_SOP000387, https://www.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Comprehensive-Exploit-Prevention-wpna.pdf.)





(*See* WBR_SOP000566, https://secure2.sophos.com/it-it/medialibrary/PDFs/other/end-of-ransomware/MarkLomanSophosInterceptX.ashx.)

260. The Accused Products perform a method that includes *invoking one of the plurality of functions as a result of receiving a call from an application programming instance*. For example,

the Accused Products monitor "sensitive API functions—like VirtualAlloc and CreateProcess…invoked by the CALL instruction."

> During normal control-flow, sensitive API functions – like VirtualAlloc and CreateProcess – are invoked by the CALL instruction. Upon invoking a sensitive API, typical ROP defenses stop code execution to determine the API invoking address, using the 'return' address which is located on top of the stack. If the instruction of the API invoking address is not a CALL, the process is terminated.

(*See* WBR_SOP000387, https://www.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Comprehensive-Exploit-Prevention-wpna.pdf.)

261. On information and belief, the Accused Products perform a method that includes *executing stack walk processing upon the invocation of one of the plurality of functions in the monitored memory space*. For example, as shown above, the Accused Products evaluate and trace a stack "[u]pon invoking a sensitive API…to determine the API invoking address, using the 'return' address which is located on top of the stack." In another example, the Accused Products include the "CallerCheck" anti-exploit module for "[p]revent[ing] API invocation from stack memory." (*See* WBR_SOP000444, https://news.sophos.com/en-us/2021/03/04/covert-code-faces-a-heap-of-trouble-in-memory/.)

262. In another example, the Accused Products "[l]ist detected IoCs mapped to the MITRE ATT&CK [Adversarial Tactics, Techniques and Common Knowledge] framework." Furthermore, the MITRE ATT&CK framework includes companion project D3FEND for defensive cybersecurity techniques, which includes "Memory Boundary Tracking" defined as "[a]nalyzing a call stack for return addresses which point to unexpected memory locations." The Accused Products incorporate the MITRE D3FEND defensive cybersecurity techniques including "Memory Boundary Tracking."

Memory Boundary Tracking

ID: D3-MBT (Memory Boundary Tracking)

Definition

Analyzing a call stack for return addresses which point to unexpected memory locations.

How it works

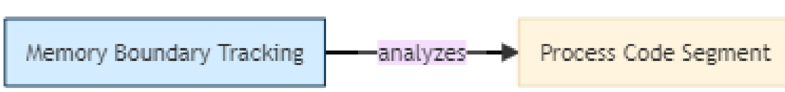
This technique monitors for indicators of whether a return address is outside memory previously allocated for an object (*i.e*. function, module, process, or thread). If so, code that the return address points to is treated as malicious code.

Considerations

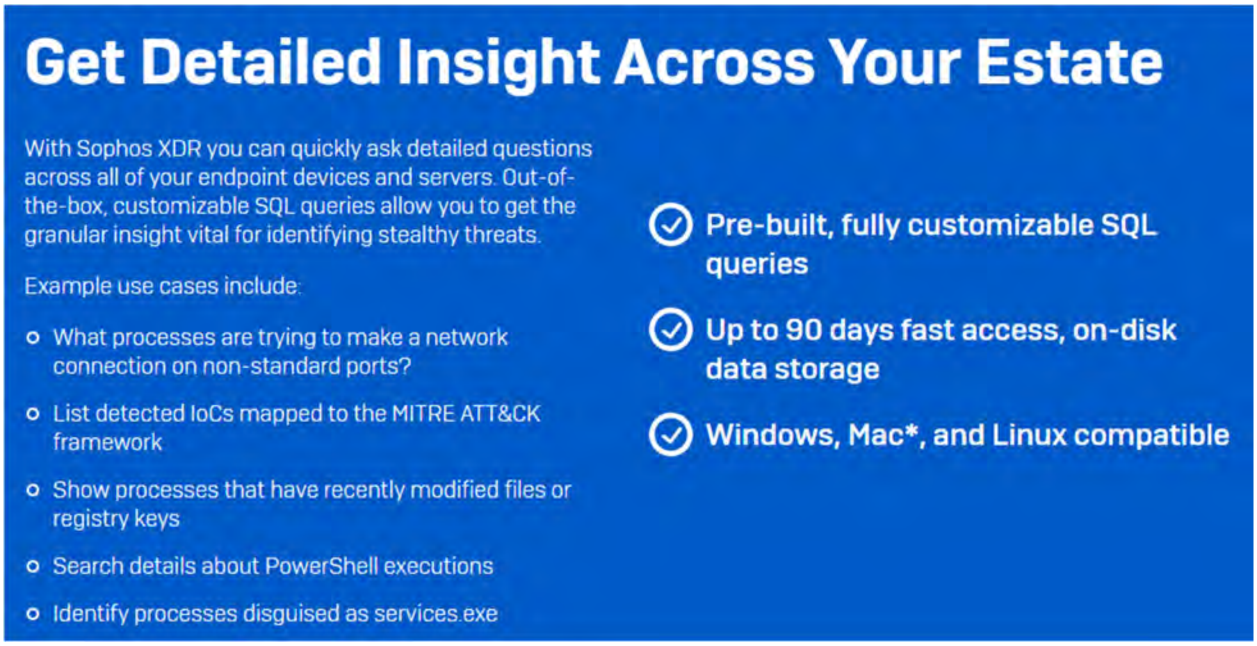
Kernel malware can manipulate memory contents, for example modifying pointers to hide processes, and thereby impact the accuracy of memory allocation information used to perform the analysis.

Digital Artifact Relationships:

This countermeasure technique is related to specific digital artifacts. Click the artifact node for more information.

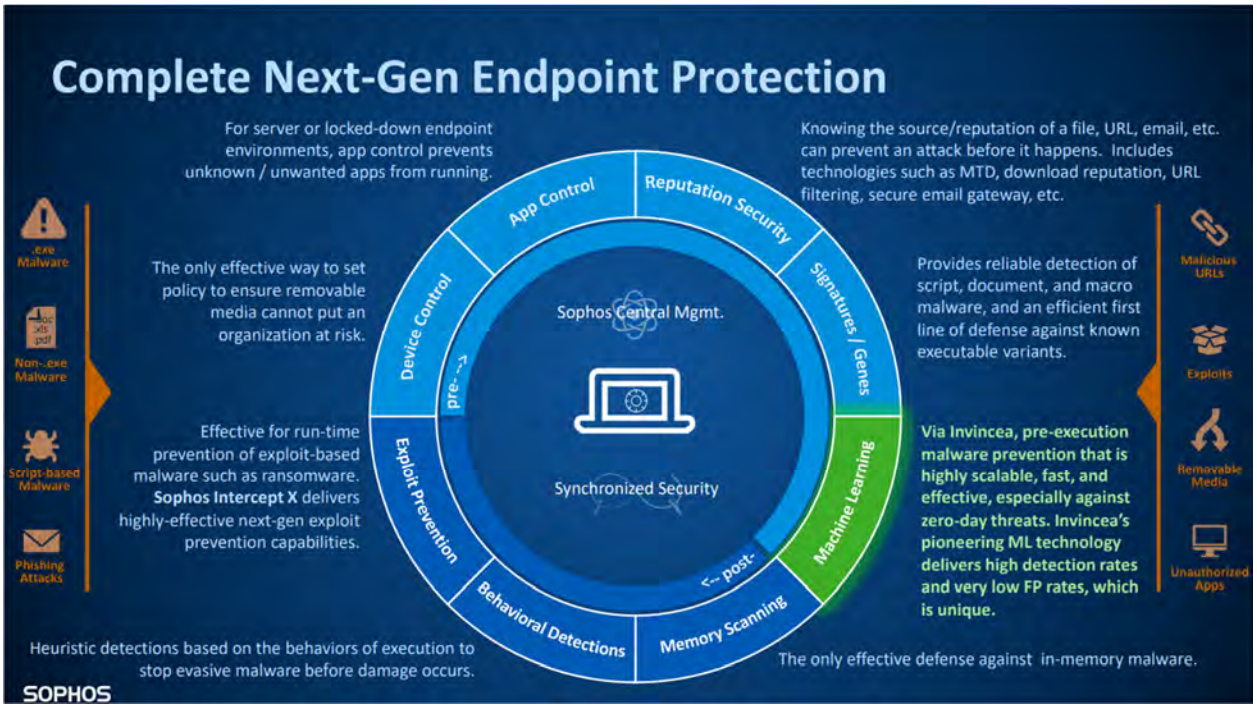


(*See* WBR_SOP000464, https://d3fend.mitre.org/technique/d3f:MemoryBoundaryTracking; *see also* WBR_SOP000467, https://www.csoonline.com/article/3625470/mitre-d3fend-explained-a-new-knowledge-graph-for-cybersecurity-defenders.html; WBR_SOP000473, https://d3fend.mitre.org/resources/D3FEND.pdf.)



(*See* WBR_SOP000659, https://www.sophos.com/en-us/content/threat-hunting.aspx.)

263. The Accused Products perform a method that includes *performing, during the executing of the stack walk processing before an address of an originating caller function is reached, a memory check for a plurality of stack entries identified during the stack walk processing to detect suspicious behavior*. For example, the Accused Products include "Memory Scanning," Behavioral Detections," and "Exploit Prevention…[e]ffective for run-time prevention of exploit-based malware."



(*See* WBR_SOP000566, https://secure2.sophos.com/it-it/medialibrary/PDFs/other/end-of-ransomware/MarkLomanSophosInterceptX.ashx.)

264. As shown above, the Accused Products include the "CallerCheck" anti-exploit module for "[p]revent[ing] API invocation from stack memory." In addition, as shown above, the Accused Products utilize the threat-based MITRE ATT&CK framework, and on information and belief, utilize companion project D3FEND for defensive cybersecurity techniques including "Memory Boundary Tracking" defined as "[a]nalyzing a call stack for return addresses which point to unexpected memory locations." (*See* WBR_SOP000444, https://news.sophos.com/en-us/2021/03/04/covert-code-faces-a-heap-of-trouble-in-memory/.)

Memory Boundary Tracking

ID: D3-MBT (Memory Boundary Tracking)

Definition

Analyzing a call stack for return addresses which point to unexpected memory locations.

How it works

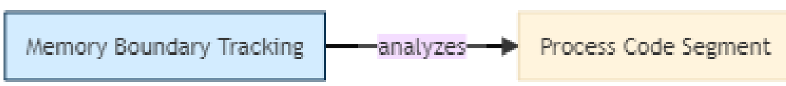
This technique monitors for indicators of whether a return address is outside memory previously allocated for an object (*i.e*. function, module, process, or thread). If so, code that the return address points to is treated as malicious code.

Considerations

Kernel malware can manipulate memory contents, for example modifying pointers to hide processes, and thereby impact the accuracy of memory allocation information used to perform the analysis.

Digital Artifact Relationships:

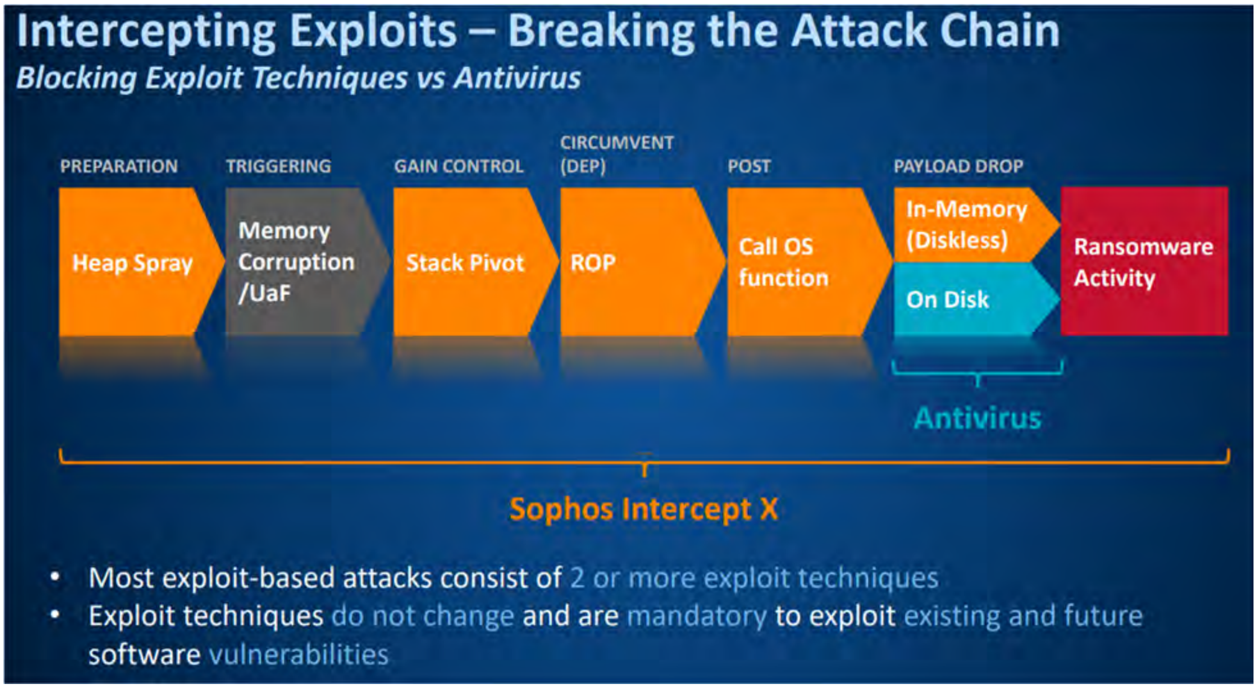
This countermeasure technique is related to specific digital artifacts. Click the artifact node for more information.



(*See* WBR_SOP000464, https://d3fend.mitre.org/technique/d3f:MemoryBoundaryTracking; *see also* WBR_SOP000467, https://www.csoonline.com/article/3625470/mitre-d3fend-explained-a-new-knowledge-graph-for-cybersecurity-defenders.html; WBR_SOP000473, https://d3fend.mitre.org/resources/D3FEND.pdf); WBR_SOP000387, https://www.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Comprehensive-Exploit-Prevention-wpna.pdf.)

265. On information and belief, the Accused Products perform a method that includes *wherein an alert of suspicious behavior is triggered when the performing of the memory check detects at least one of the following: code execution is attempted from non-executable memory, a base pointer is identified as being invalid, an invalid stack return address is identified, attempted*

execution of a return-oriented programming technique is detected, the base pointer is detected as being outside a current thread stack, and a return address is detected as being inside a virtual memory area. For example, the Accused Products detect and prevent malware memory exploitations including "Stack Pivot," "Stack Exec," "Stack-based ROP [return-oriented programming] Mitigations," and "Shellcode."



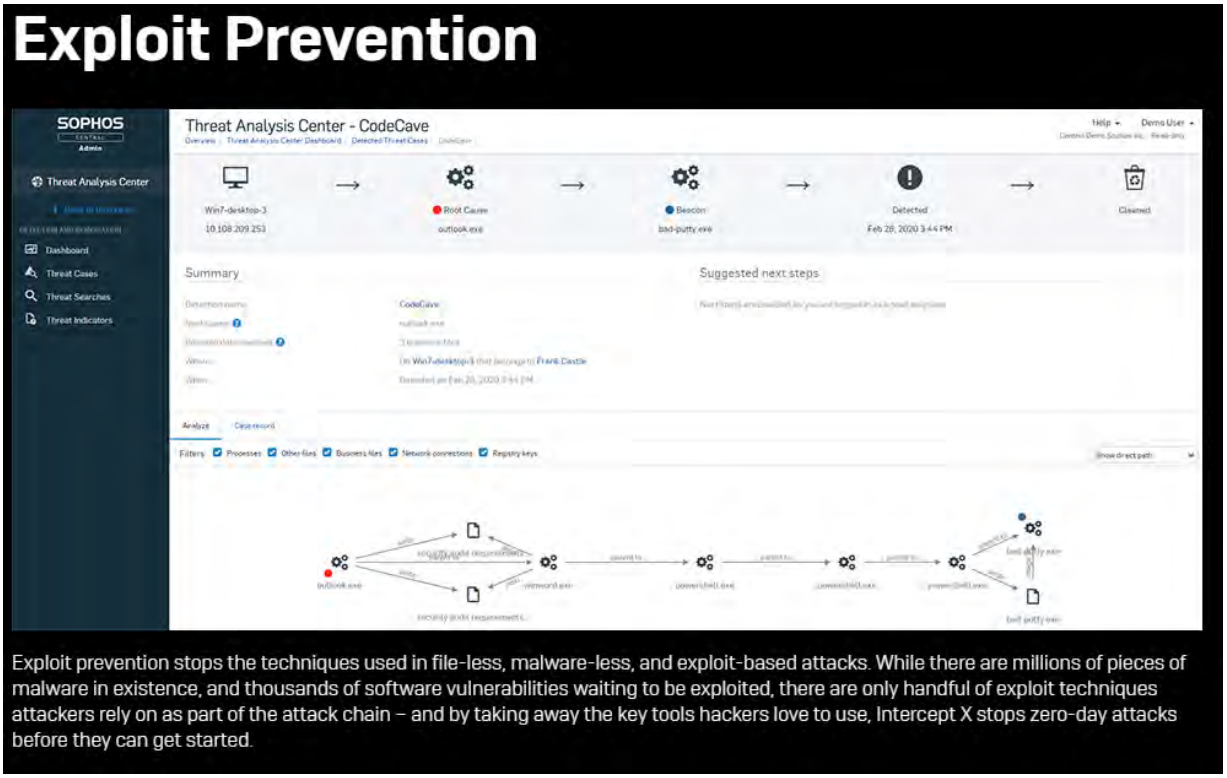
(*See* WBR_SOP000566, https://secure2.sophos.com/it-it/medialibrary/PDFs/other/end-of-ransomware/MarkLomanSophosInterceptX.ashx.)

266. In another example, the Accused Products prevent malware attacks using stack memory including "DEP," "ROP," "CallerCheck," "StackPivot," "StackExec," and "AmsiGuard."

Description	Module	Level	Equivalent in Windows 10
Enforce Data Execution Prevention (DEP)	DEP	Application	Yes
Mandatory ASLR on modules	DEP (ASLR)	Application	Yes
Bottom-up ASLR	DEP (ASLR)	Application	Yes
Validate exception chains	SEHOP	Application	Yes
Validate API invocation	ROP	Application	Optional
Prevent API invocation from stack memory	CallerCheck	Application	–
Prevent process creation from dynamic memory	CallerCheck	Application	–
Import address filtering (IAF)	IAF	Application	Optional
Validate stack integrity	StackPivot	Application	Optional
Validate stack memory protection	StackExec	Application	Optional
Prevent in-memory manipulation of AMSI.DLL	AmsiGuard	System	

(*See* WBR_SOP000444, https://news.sophos.com/en-us/2021/03/04/covert-code-faces-a-heap-of-trouble-in-memory/.)

267. The Accused Products perform a method that includes *wherein when an alert of suspicious behavior is triggered, preventing execution of a payload for the invoked function from operating*. For example, the Accused Products "stop[] the techniques used in file-less, malware-less, and exploit-based attacks."



(*See* WBR_SOP000222, https://www.sophos.com/en-us/products/endpoint-antivirus.aspx.)

268. In another example, the Accused Products are demonstrated preventing shellcode execution by utilizing non-executable memory and using "Data Execution Prevention…to prevent abuse of a buffer overflow."

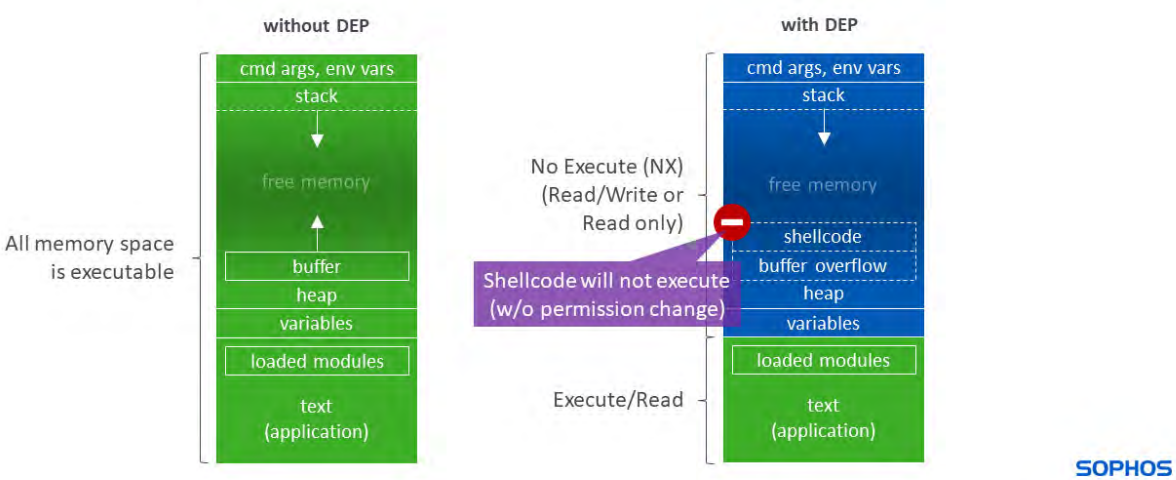


Figure 5 – Data Execution Prevention helps to prevent abuse of a buffer overflow

(*See* WBR_SOP000444, https://news.sophos.com/en-us/2021/03/04/covert-code-faces-a-heap-of-trouble-in-memory/.)

269. Each claim in the ’591 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ’591 Patent.

270. Defendant has been aware of the ’591 Patent since at least the filing of this Complaint. Further, Plaintiffs have marked its products with the ’591 Patent, including on its web site, since at least July 2020.

271. Defendant directly infringes at least claim 1 of the ’591 Patent, literally or under the doctrine of equivalents, by performing the steps described above. For example, on information and belief, Defendant performs the claimed method as described above by running this software and corresponding systems to protect its own computer and network operations. On information and belief, Defendant also performs the claimed method as described above when testing the operation of the Accused Products and corresponding systems. As another example, Defendant performs the claimed method as described above when providing or administering services to third parties, customers, and partners using the Accused Products.

272. Defendant’s partners, customers, and end users of the Accused Products and corresponding systems and services directly infringe at least claim 1 of the ’591 Patent, literally or under the doctrine of equivalents, at least by using the Accused Products and corresponding systems and services, as described above.

273. Defendant has actively induced and is actively inducing infringement of at least claim 1 of the ’591 Patent with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b). For example, Defendant encourages and induces customers with specific intent to use Sophos’s security

software in a manner that infringes claim 1 of the '591 Patent at least by offering and providing software that performs a method that infringes claim 1 when installed and operated by the customer, and by engaging in activities relating to selling, marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including the activities described below.

274. Defendant encourages, instructs, directs, and/or requires third parties—including its certified partners and/or customers to perform the claimed method using the software, services, and systems in infringing ways, as described above.

275. Defendant further encourages and induces its customers to infringe claim 1 of the '591 Patent: 1) by making its security services available on its website, providing applications that allow users to access those services, widely advertising those services, and providing technical support and instructions to users, and 2) through activities relating to marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including its Sophos security software, and services in the United States. (*See* WBR_SOP000506, https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx; WBR_SOP000525, https://partners.sophos.com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1; WBR_SOP000209, https://secure2.sophos.com/en-us/security-news-trends/whitepapers/gated-wp/cybersecurity-system-buyers-uide.aspx%23form Frame.)

276. For example, on information and belief, Defendant shares instructions, guides, and manuals, which advertise and instruct third parties on how to use the software as described above, including with at least customers and partners. (*Id.*) On further information and belief, Defendant also provides customer service and technical support to purchasers of the Accused Products and corresponding system and services, which directs and encourages customers to perform certain

actions that use the Accused Products in an infringing manner. (*Id*.)

277. Defendant and/or its partners recommend and sell the Accused Products and provide technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Defendant and/or one of its partners, which obligates each customer to perform certain actions in order to use the Accused Products. Further, in order to receive the benefit of Defendant's and/or its partner's continued technical support and their specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that performs the claimed method and infringes the '591 Patent.

278. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Product remains operational for each customer through ongoing technical support, on information and belief, Defendant and/or its partners affirmatively aid and abet each customer's use of the Accused Products in a manner that performs the claimed method of, and infringes, the '591 Patent.

279. Defendant also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the claimed methods, and not a staple article or commodity of commerce suitable for substantial non-infringing uses. Indeed, as shown above, the Accused Products and the example functionality described below have no substantial non-infringing uses but are specifically designed to practice the '591 Patent.

280. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the anti-exploit, memory analyzing techniques, and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. For example, without scanning the memory within which a malicious object executes and ensuring that it does not exceed its permissible memory boundary, the Accused Products could not detect when that object had attempted to engage in an illicit behavior, and thus could not prevent that behavior. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’591 Patent, that functionality could not be performed.

281. Additionally, the accused functionality, including the anti-exploit, memory analyzing techniques, and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without scanning the memory in which an object executes and preventing the object from exceeding its memory boundary, the Accused Products could not assess when the object was about to engage in prohibited behavior and prevent it from doing so. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’591 Patent, that functionality could not be performed.

282. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Defendant constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionalities (including the anti-exploit, memory analyzing techniques) constitute a material part of the inventions claimed at least because such functionality is integral to the processes identified above (such as "monitoring a memory space of a process for execution of at least one monitored function…"; "invoking one of the plurality of functions…"; "executing stack walk processing…"; "performing, during the executing of the stack walk processing before an address of an originating caller function is reached, a memory check…"), as recited in the claims of the '591 Patent. None of these products are staple goods--they are sophisticated and customized cyber security and malware detection products, methods, and systems.

283. Defendant's infringing actions have continued after the original Complaint was filed. Defendant had knowledge of the '591 Patent and of the specific conduct that constituted infringement of the '591 Patent at least based on the original Complaint, but has continued to engage in its infringing activity, including by selling, making, configuring, and installing the Accused Products and performing the accused functionality, and by engaging in activities that constitute inducing infringement and contributory infringement as described above.

284. On information and belief, the infringing actions of each partner, customer, and/or end-user of the Accused Products are attributable to Defendant. For example, on information and belief, Defendant directs and controls the activities or actions of its partners or others in connection with the Accused Products by contractual agreement or otherwise requiring partners or others to provide information and instructions to customers who acquire the Accused Products which, when

followed, results in infringement. Defendant further directs and controls the operation of devices executing the Accused Products by programming the software which, when executed by a customer or end user, perform the method steps of at least claim 1 of the ’591 Patent.

285. Plaintiffs have suffered and continue to suffer damages, including lost profits, as a result of Defendant’s infringement of the ’591 Patent. Defendant is therefore liable to Plaintiffs under 35 U.S.C. § 284 for damages in an amount that adequately compensates Plaintiffs for Defendant’s infringement, but no less than a reasonable royalty.

286. Plaintiffs will continue to suffer irreparable harm unless this Court preliminarily and permanently enjoins Defendant, its agents, employees, representatives, and all others acting in concert with Defendant from infringing the ’591 Patent.

287. Defendant’s infringement of the ’591 Patent is knowing and willful. Defendant had actual knowledge of the ’591 Patent at least by the time Plaintiffs filed this lawsuit and had constructive knowledge of the ’591 Patent from at least the date Plaintiffs marked their products with the ’591 Patent and/or provided notice of the ’591 Patent on their website.

288. On information and belief, despite Defendant’s knowledge of the Asserted Patents and Plaintiffs’ patented technology, Defendant made the deliberate decision to sell products and services that they knew infringe the ’591 Patent. Defendant’s continued infringement of the ’591 Patent with knowledge of the ’591 Patent constitutes willful infringement.

289. Plaintiffs’ allegations of infringement, indirect infringement, and willful infringement with respect to this patent are further set forth in Plaintiffs’ Disclosure of Asserted Claims and Preliminary Infringement Contentions Pursuant to the Court’s Standing Order Governing Patent Proceedings served July 12, 2022.

SIXTH CAUSE OF ACTION
(INFRINGEMENT OF THE ’844 PATENT)

290. Plaintiffs reallege and incorporate by reference the allegations of the preceding paragraphs of this Complaint.

291. Defendant has infringed and continues to infringe one or more claims of the ’844 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including features such as Intercept X Advanced with EDR (“Intercept X”), at least when used for their ordinary and customary purposes, practice each element of at least claim 1 of the ’844 Patent, as described below.

292. Claim 1 of the ’844 Patent recites:

> 1. A computer-implemented method comprising:
>
> extracting a plurality of static data points from an executable file without decrypting or unpacking the executable file, wherein the plurality of static data points represent predefined character strings in the executable file;
>
> generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points based on a collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files, wherein the collection of data comprises at least a portion of the plurality of static data points, and
>
> wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more static data points from the plurality of extracted static data points is within a predetermined range; and
>
> evaluating the feature vector using support vector processing to determine whether the executable file is harmful.

293. The Accused Products perform each element of the method of claim 1 of the ’844 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform *a computer-implemented method*, as further explained below. For example, Intercept X employs a

"deep neural network…trained on hundreds of millions of samples to detect when a file is malicious, potentially unwanted, or legitimate."

> **OXFORD, U.K. – Nov. 2, 2017** Sophos (LSE:SOPH), a global leader in network and endpoint security today announced that deep learning driven malware detection is now available through its Intercept X early access program. This deep learning capability has been developed using technology from Invincea, acquired by Sophos in February 2017.
>
> First released in September 2016, Sophos Intercept X is a next-generation endpoint security product that stops zero-day malware, blocks all exploit techniques known today and includes an advanced anti-ransomware feature that can stop both known and unknown ransomware variants within seconds. Deployed through the cloud-based management platform Sophos Central, Intercept X can be installed alongside existing endpoint security software from any vendor, immediately boosting endpoint protection by stopping malicious code before it can do harm.
>
> Deep learning is a branch of machine learning and artificial intelligence that leverages an artificial neural network to build a model used to make predictions with speed, scale, and judgement that exceed human capabilities. The deep neural network of Intercept X is trained on hundreds of millions of samples to detect when a file is malicious, potentially unwanted, or legitimate. Deep learning is more effective than traditional machine learning approaches because of its larger scale training set, smaller model, and more effective detections.

(*See* WBR_SOP000657, https://www.sophos.com/en-us/press-office/press-releases/2017/11/sophos-adds-deep-learning-capabilities-to-intercept-x-early-access-program.aspx.)

294. The Accused Products perform a method of *extracting a plurality of static data points from an executable file without decrypting or unpacking the executable file, wherein the plurality of static data points represent predefined character strings in the executable file*. For example, Intercept X's "Deep Learning" technology, "extract[s] millions of features from a file and determine if it is malicious before the program executes." These extracted features include the most "significant strings found in the file," the attribute "Findcrypt," which "shows any suspicious cryptographic constants" found within the file, and the attribute "Resources," which "specifies a resource" within the file "that appears to be compressed or encrypted."

Deep Learning Malware Detection

With the new deep learning model, we are able to perform a signatureless pre-execution evaluation of any executable file and determine if it is malware, potentially unwanted software, or a legitimate application.

At Sophos we've taken a unique approach to our security machine learning capabilities: we've invested heavily in deep neural network technology over more prevalent methods that, while still dominant in the security industry, are being rapidly abandoned by the machine learning computer science community.

How does Intercept X detect malicious executable files?

Instead of performing a signature and heuristic scan as traditional antivirus does, deep neural networks are able to extract millions of features from a file and determine if it is malicious before the program executes. The deep learning model learns what to look for in the code, how adversaries evade detection, how they build their software, and how the software plans to deploy and run. This information is evaluated by a multi-stage deep learning algorithm to determine how similar the software is to malware or potentially unwanted software. Depending on the score it is classified as malicious, potentially unwanted, or legitimate. It does all of this in about 20 milliseconds with a model that is under 20MB in size.

(*See* WBR_SOP000282, https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

Machine learning analysis

Under **Machine learning analysis**, you can see full results of our analysis.

Attributes shows a comparison of the file's attributes with those in millions of known bad and known good files. This enables you to determine how suspicious each attribute is and therefore whether the file is likely to be good or bad. You may see the following attributes:

- **Imports** describes the functionality that the file uses from external DLLs.
- **Strings** describes the most significant strings in the file.
- **Compilers** specifies what was used to compile the source code, for example C++, Delphi, Visual Basic, .NET.
- **Mitigation** describes techniques that the file uses to avoid being exploited.
- **Resources** specifies a resource that seems to be compressed or encrypted.
- **Summary** often relates to build or compilation dates, for example.

- **Packer** often specifies something of note about a specific section of the file, for example a suspicious section name or the fact that a section is both writable and executable.
- **Peid** refers to the output of PEiD, a third-party tool that scans a PE file against various malware signatures.
- **Btcaddress** shows any valid Bitcoin address that is found in the file.
- **Findcrypt** shows any suspicious cryptographic constants.

Code similarity shows a comparison of the file with millions of known bad and known good files, and lists the closest matches. Other matches count toward the result and may affect the rating for the file. The more bad files the file matches and the more closely it matches them, the more suspicious the file is.

File/path shows a comparison of the file's path with that of millions of known bad and known good files. If the file's path more closely matches the path of known bad files, the file is more likely to be suspicious. The path and file name used for comparison are either yours (if you requested the latest intelligence) or those from the last customer who sent us a file. We hide sensitive information in other customers' paths.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html.)

295. The Accused Products perform a method of *generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points based on a collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files, wherein the collection of data comprises at least a portion of the plurality of static data points*. For example, Intercept X employs "Deep Learning models" (*i.e.*, advanced machine learning models) that have been trained on "hundreds of millions of samples" to classify the features extracted from a file to "determine if a file is benign or malicious…before the file executes." Such samples include "Potentially Unwanted Applications" such as adware. Before using a deep learning model to classify the features extracted from a file, Intercept X creates a "vector of information" that translates those features into data that the model can "intake" and "process."

Performance: Sophos' Deep Learning technology is incredibly fast. In less than 20 milliseconds the model is able to extract millions of features from a file, conduct a deep analysis, and determine if a file is benign or malicious. This entire process happens before the file executes.

SophosLabs: One of the most important aspects to any model is the data that used for training. Our team of data scientists are part of the SophosLabs group, granting them access to hundreds of millions of samples. This allows them to create the best possible predictions in our models. The integration between the two groups also leads to better data labeling (and therefore better modeling). The bi-directional sharing of threat intelligence and real-world feedback between the team of data scientists and threat researchers continuously improves the accuracy of our models.

(*See* WBR_SOP000300, https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophos-intercept-x-deep-learning-dsna.pdf.)

Feature Engineering in Machine Learning

Before creating a machine learning model, it's important to prepare our data. Preparing the data requires translating it into a language our model can understand. This is referred to as feature engineering.

Artificial neural network models intake data as a vector of information, so simply feeding the model a URL – which is not in the language of a vector – means that the model can't process it without some manipulation. There are countless ways that samples can be translated into features, though it takes some domain knowledge to do so. Using the URL example again, one way to translate a URL into a usable language is through a combination of ngramming and hashing. Ngrams are a popular method in DNA sequencing research. For example, the results of a three-gram ngram for the URL "https://sophos.com/company/careers.aspx" would be:

['htt', 'ttp', 'tps', 'ps:', 's:/', '://', '//s', '/so', 'sop', 'oph', 'pho', 'hos', 'os.', 's.c', '.co', 'com', 'om/', 'm/c', '/co', 'com', 'omp', 'mpa', 'pan', 'any', 'ny/', 'y/c', '/ca', 'car', 'are', 'ree', 'eer', 'ers', 'rs.', 's.a', '.as', 'asp', 'spx']

Once the ngrams are calculated, we need to translate them into a numerical representation. This can be done through a hashing mechanism. We will create an n-length long vector – say 1000 – and hash each ngram using a hashing algorithm. The resulting number from the hash of a particular ngram will be the index of which we will add 1. For example, if the first ngram 'htt' results in a hash of three and our vector is five units long, the result would be [0, 0, 1, 0, 0]. We continue this process for every ngram and for every URL until we have the list of URLs completely transformed into individual n-length vectors. When using this method for our toy model, these vectors will be 1,000 units long.

(*See* WBR_SOP000302, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/machine-learning-how-to-build-a-better-threat-detection-model.pdf?cmp=70130000001xKqzAAE.)

2.1 MLP Model

An MLP is a class of feedforward ANN with an input layer, an output layer, and one or more hidden layers between them. Each node in one layer fully connects to every node in the following layer. Except for nodes in the input layer, each node is a neuron (or processing element) with a nonlinear activation function associated with a scalar weight which is adjusted during training. An MLP is a supervised learning algorithm that learns to identify data patterns for classification or regression. In order to carry out the learning process, one needs to extract a digital representation $\boldsymbol{x}$ of a given object or event that needs to be fed into the MLP. The learning task becomes to find a multidimensional function $\Phi(\cdot)$ between input $\boldsymbol{x}$ and target $\boldsymbol{y}$

$$\boldsymbol{y} \cong \Phi(\boldsymbol{x}) \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^N$, a real-valued input feature vector $\boldsymbol{x} = [x_1, \dots, x_N]^T$ in an N dimensional feature space, with $(\cdot)^T$ denoting the transpose operation. Similarly, $\boldsymbol{y} \in \mathbb{R}^M$, a real-valued target classification vector $\boldsymbol{y} = [y_1, \dots, y_M]^T$ in an M dimensional classification space. In other words, an MLP learning process is to find $\Phi(\cdot)$

(*See* WBR_SOP000318, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophos-black-hat-2018-technicalpaper.pdf.)

Generic ML PUA detections

Potentially Unwanted Application (PUA) is a term used to describe applications that, while not malicious, are generally considered unsuitable for business networks.

A Generic ML PUA detection is generated by Sophos Intercept X's Machine Learning (ML) engine, also referred to by the specific Sophos approach Deep Learning and is designed to detect PUAs in PE (Portable Executable) files such as:

- .exe
- .sys
- .dll
- .pif
- .scr
- and many more

If a detection of this type has been received, it is because Sophos has detected a file on the computer that our Deep Learning threat model has decided is a PUA. This is a pre-execution detection meaning the file was detected before it was able to be run.

Major PUA classifications

The major PUA classifications are:

- adware
- dialer
- non-malicious spyware
- remote administration tools
- hacking tools

(*See* WBR_SOP000318, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophos-black-hat-2018-technicalpaper.pdf.)

296. The feature vector is "evaluated by a multi-stage deep learning algorithm to determine how similar the software is to malware or potentially unwanted software. Depending on the score [the file] is classified as malicious, potentially unwanted, or legitimate."

Deep Learning Malware Detection

With the new deep learning model, we are able to perform a signatureless pre-execution evaluation of any executable file and determine if it is malware, potentially unwanted software, or a legitimate application.

At Sophos we've taken a unique approach to our security machine learning capabilities: we've invested heavily in deep neural network technology over more prevalent methods that, while still dominant in the security industry, are being rapidly abandoned by the machine learning computer science community.

How does Intercept X detect malicious executable files?

Instead of performing a signature and heuristic scan as traditional antivirus does, deep neural networks are able to extract millions of features from a file and determine if it is malicious before the program executes. The deep learning model learns what to look for in the code, how adversaries evade detection, how they build their software, and how the software plans to deploy and run. This information is evaluated by a multi-stage deep learning algorithm to determine how similar the software is to malware or potentially unwanted software. Depending on the score it is classified as malicious, potentially unwanted, or legitimate. It does all of this in about 20 milliseconds with a model that is under 20MB in size.

(*See* WBR_SOP000282, https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

297. The Accused Products perform a method *wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more static data points from the plurality of extracted static data points is within a predetermined range*. For example, Intercept X employs "feature selection" to only "keep relevant features before feeding them into" its deep learning models, "identifying and removing as much noisy and redundant information as possible from extracted features."

A crucial step in an ML workflow is feature extraction, which can be hand-crafted-based on human expertise, or automatically learned by training modern deep learning models such as convolutional neural networks (CNNs). It is natural to believe that more extracted features can provide better characterization of a learning task and more discriminating power. However, increasing the dimension of the feature vector could result in feature redundancy and noise. Redundant and irrelevant features can cause performance deterioration of an ML model with overfitting and generalization problems. Additionally, excessively increased number of features could significantly slow down a learning process. Therefore, it is of fundamental importance to only keep relevant features before feeding them into an ML model, which leads to requiring feature selection (or feature dimensionality reduction). Feature selection can be seen as the process of identifying and removing as much noisy and redundant information as possible from extracted features.

(*See* WBR_SOP000329, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophoslabs-machine-learning-tp.pdf.)

298. The Accused Products perform a method of *evaluating the feature vector using support vector processing to determine whether the executable file is harmful*. For example, and as explained above, Intercept X employs "Deep Learning models" (*i.e.*, advanced machine learning models) that have been trained on "hundreds of millions of samples" to classify the features extracted from a file to "determine if a file is benign or malicious…before the file executes." Before using a deep learning model to classify the features extracted from a file, Intercept X creates a "vector of information" that translates those features into data that the model can "intake" and "process."

Performance: Sophos' Deep Learning technology is incredibly fast. In less than 20 milliseconds the model is able to extract millions of features from a file, conduct a deep analysis, and determine if a file is benign or malicious. This entire process happens before the file executes.

SophosLabs: One of the most important aspects to any model is the data that used for training. Our team of data scientists are part of the SophosLabs group, granting them access to hundreds of millions of samples. This allows them to create the best possible predictions in our models. The integration between the two groups also leads to better data labeling [and therefore better modeling]. The bi-directional sharing of threat intelligence and real-world feedback between the team of data scientists and threat researchers continuously improves the accuracy of our models.

(*See* WBR_SOP000300, https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophos-intercept-x-deep-learning-dsna.pdf.)

Feature Engineering in Machine Learning

Before creating a machine learning model, it's important to prepare our data. Preparing the data requires translating it into a language our model can understand. This is referred to as feature engineering.

Artificial neural network models intake data as a vector of information, so simply feeding the model a URL – which is not in the language of a vector – means that the model can't process it without some manipulation. There are countless ways that samples can be translated into features, though it takes some domain knowledge to do so. Using the URL example again, one way to translate a URL into a usable language is through a combination of ngramming and hashing. Ngrams are a popular method in DNA sequencing research. For example, the results of a three-gram ngram for the URL "https://sophos.com/company/careers.aspx" would be:

['htt', 'ttp', 'tps', 'ps:', 's:/', '://', '//s', '/so', 'sop', 'oph', 'pho', 'hos', 'os.', 's.c', '.co', 'com', 'om/', 'm/c', '/co', 'com', 'omp', 'mpa', 'pan', 'any', 'ny/', 'y/c', '/ca', 'car', 'are', 'ree', 'eer', 'ers', 'rs.', 's.a', '.as', 'asp', 'spx']

Once the ngrams are calculated, we need to translate them into a numerical representation. This can be done through a hashing mechanism. We will create an n-length long vector – say 1000 – and hash each ngram using a hashing algorithm. The resulting number from the hash of a particular ngram will be the index of which we will add 1. For example, if the first ngram 'htt' results in a hash of three and our vector is five units long, the result would be [0, 0, 1, 0, 0]. We continue this process for every ngram and for every URL until we have the list of URLs completely transformed into individual n-length vectors. When using this method for our toy model, these vectors will be 1,000 units long.

(*See* WBR_SOP000302, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/machine-learning-how-to-build-a-better-threat-detection-model.pdf?cmp=70130000001xKqzAAE.)

299. The feature vector is "evaluated by a multi-stage deep learning algorithm to determine how similar the software is to malware or potentially unwanted software. Depending on the score [the file] is classified as malicious, potentially unwanted, or legitimate." That evaluation is processed using support vector processing.

Deep Learning Malware Detection

With the new deep learning model, we are able to perform a signatureless pre-execution evaluation of any executable file and determine if it is malware, potentially unwanted software, or a legitimate application.

At Sophos we've taken a unique approach to our security machine learning capabilities: we've invested heavily in deep neural network technology over more prevalent methods that, while still dominant in the security industry, are being rapidly abandoned by the machine learning computer science community.

How does Intercept X detect malicious executable files?

Instead of performing a signature and heuristic scan as traditional antivirus does, deep neural networks are able to extract millions of features from a file and determine if it is malicious before the program executes. The deep learning model learns what to look for in the code, how adversaries evade detection, how they build their software, and how the software plans to deploy and run. This information is evaluated by a multi-stage deep learning algorithm to determine how similar the software is to malware or potentially unwanted software. Depending on the score it is classified as malicious, potentially unwanted, or legitimate. It does all of this in about 20 milliseconds with a model that is under 20MB in size.

(*See* WBR_SOP000329, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophoslabs-machine-learning-tp.pdf.)

300. Each claim in the ’844 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ‘844 Patent.

301. Defendant has been aware of the ’844 Patent since at least the filing of this Complaint. Further, Plaintiffs have marked its products with the ’844 Patent, including on its web site, since at least July 2020.

302. Defendant directly infringes at least claim 1 of the ’844 Patent, either literally or under the doctrine of equivalents, by performing the steps described above. For example, on information and belief, Defendant performs the claimed method as described above by running the

Sophos security software and corresponding systems to protect its own computer and network operations. On information and belief, Defendant also performs the claimed method as described above when testing the operation of the Accused Products and corresponding systems. As another example, Defendant performs the claimed method as described above when providing or administering services to third parties, customers, and partners using the Accused Products.

303. Defendant's partners, customers, and end users of the Accused Products and corresponding systems and services directly infringe at least claim 1 of the '844 Patent, literally or under the doctrine of equivalents, at least by using the Accused Products and corresponding systems and services, as described above.

304. Defendant has actively induced and is actively inducing infringement of at least claim 1 of the '844 Patent with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b). For example, Sophos encourages and induces customers with specific intent to use Sophos's security software in a manner that infringes claim 1 of the '844 Patent at least by offering and providing software that performs a method that infringes claim 1 when installed and operated by the customer, and by engaging in activities relating to selling, marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including the activities described below.

305. Defendant encourages, instructs, directs, and/or requires third parties—including its certified partners and/or customers to perform the claimed method using the software, services, and systems in infringing ways, as described above.

306. Defendant further encourages and induces its customers to infringe claim 1 of the '844 Patent: 1) by making its security services available on its website, providing applications that allow users to access those services, widely advertising those services, and providing technical

support and instructions to users, and 2) through activities relating to marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including its Sophos security software, and services in the United States. (*See* https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx; WBR_SOP000525, https://partners.sophos.com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1; WBR_SOP000209, https://secure2.sophos.com/en-us/security-news-trends/whitepapers/gated-wp/cybersecurity-system-buyers-guide.aspx%23formFrame; *see also* WBR_SOP000214, https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophos-synchronized-security-ds.pdf.)

307. For example, on information and belief, Defendant shares instructions, guides, and manuals, which advertise and instruct third parties on how to use the software as described above, including at least customers and partners. (*Id*.) On further information and belief, Defendant provides customer service and technical support to purchasers of the Accused Products and corresponding system and services, which directs and encourages customers to perform certain actions that use the Accused Products in an infringing manner. (*Id*.)

308. Defendant and/or its partners recommend and sell the Accused Products and provide technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Defendant and/or one of its partners, which obligates each customer to perform certain actions in order to use the Accused Products. Further, in order to receive the benefit of Defendant's or its partner's continued technical support and their specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that performs the claimed method and infringes the '844

Patent.

309. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Product remains operational for each customer through ongoing technical support, on information and belief, Defendant and/or Sophos's partners affirmatively aid and abet each customer's use of the Accused Products in a manner that performs the claimed method of, and infringes, the '844 Patent.

310. Defendant also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the claimed methods, and not a staple article or commodity of commerce suitable for substantial non-infringing uses. Indeed, as shown above, the Accused Products and the example functionality described below have no substantial non-infringing uses but are specifically designed to practice the '844 Patent.

311. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the machine learning static analysis and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. For example, without extracting a set of static data points from an object, the Accused Products could not create feature vector that they could feed to their machine learning model. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system

and components identified above that practice the ’844 Patent, that functionality could not be performed.

312. Additionally, the accused functionality, including the machine learning static analysis and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without extracting and analyzing a set of static features from an object and calibrating a feature vector on the basis of that analysis, the Accused Products could not deploy their machine learning model to determine whether or not that object is malicious or benign. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’844 Patent, that functionality could not be performed.

313. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Defendant constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionality (including the machine learning static analysis) constitute a material part of the inventions claimed at least because such an attack chain is integral to the processes identified above (such as “extracting a plurality of static data points from an executable file…”; “generating a feature vector from the plurality of static data points…”; “wherein one or more features of the feature vector are selectively turned on or off…”; “evaluating the feature vector…”) as recited in the claims of the ’844 Patent. None of these products are staple goods—they are sophisticated and customized cyber security and malware

detection products, methods, and systems.

314. Defendant's infringing actions have continued after the original Complaint was filed. Defendant had knowledge of the '844 Patent and of the specific acts that constitutes infringement of the '844 Patent at least based on the original Complaint, but has continued to engage in the infringing activity, including by selling, making, configuring, and installing the Accused Products and performing the accused functionality, and by engaging in activities that constituted infringement and contributory infringement as described above.

315. On information and belief, the infringing actions of each partner, customer, and/or end-user of the Accused Products are attributable to Defendant. For example, on information and belief, Defendant directs and controls the activities or actions of its partners or others in connection with the Accused Products by contractual agreement or otherwise requiring partners or others to provide information and instructions to customers who acquire the Accused Products which, when followed, results in infringement. Defendant further directs and controls the operation of devices executing the Accused Products by programming the software which, when executed by a customer or end user, perform the method steps of at least claim 1 of the '844 Patent.

316. Plaintiffs have suffered and continue to suffer damages, including lost profits, as a result of Defendant's infringement of the '844 Patent. Defendant is therefore liable to Plaintiffs under 35 U.S.C. § 284 for damages in an amount that adequately compensates Plaintiffs for Defendant's infringement, but no less than a reasonable royalty.

317. Plaintiffs will continue to suffer irreparable harm unless this Court preliminarily and permanently enjoins Defendant, its agents, employees, representatives, and all others acting in concert with Defendant from infringing the '844 Patent.

318. Defendant’s infringement of the ’844 Patent is knowing and willful. Defendant had actual knowledge of the ’844 Patent at least by the time Plaintiffs filed this lawsuit and had constructive knowledge of the ’844 Patent from at least the date Plaintiffs marked their products with the ’844 Patent and/or provided notice of the ’844 Patent on their website.

319. On information and belief, despite Defendant’s knowledge of the Asserted Patents and Plaintiffs’ patented technology, Defendant made the deliberate decision to sell products and services that they knew infringe the ’844 Patent. Defendant’s continued infringement of the ’844 Patent with knowledge of the ’844 Patent constitutes willful infringement.

320. Plaintiffs’ allegations of infringement, indirect infringement, and willful infringement with respect to this patent are further set forth in Plaintiffs’ Disclosure of Asserted Claims and Preliminary Infringement Contentions Pursuant to the Court’s Standing Order Governing Patent Proceedings served July 12, 2022.

SEVENTH CAUSE OF ACTION
(INFRINGEMENT OF THE ’721 PATENT)

321. Plaintiffs reallege and incorporate by reference the allegations of the preceding paragraphs of this Complaint.

322. Sophos has infringed and continues to infringe one or more claims of the ’721 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including features such as Sophos Firewall, at least when used for their ordinary and customary purposes, practice each element of at least claim 1 of the ’721 Patent as described below.

323. For example, claim 1 of the ’721 Patent recites:

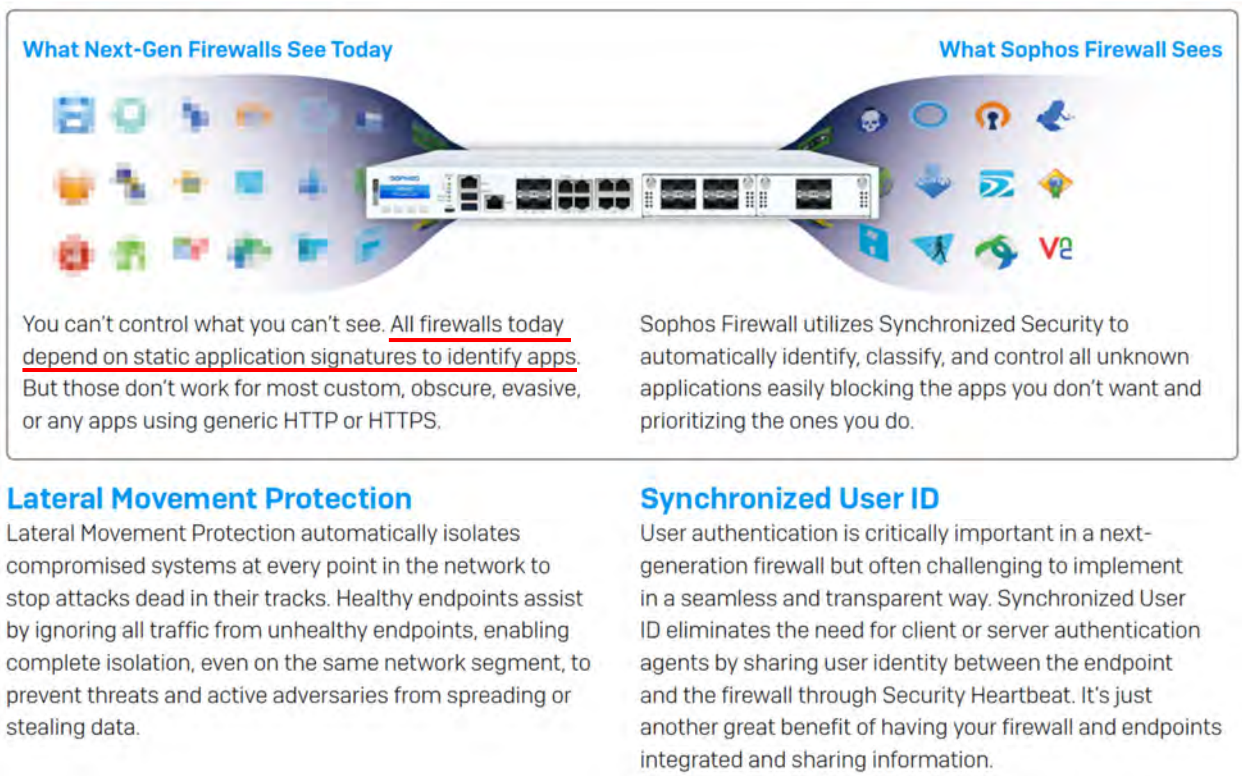
> 1. A method of classifying a computer object as malware, the method comprising:

> receiving, at a first threat server, details of a first computer object from a first remote computer, wherein the details of the first computer object include data uniquely identifying the first computer object;
>
> determining, by the first threat server, whether the first computer object has been previously seen by comparing the data uniquely identifying the first computer object to a plurality of data uniquely identifying plural computer objects in a first database associated with the first threat server;
>
> receiving additional information about the first computer object from the first remote computer when the first computer object has not been previously seen;
>
> storing the details of the first computer object and the received additional information about the first computer object in a second database associated with the first threat server when the first computer object has not been previously seen;
>
> providing contents of the second database to at least one database associated with a central server, wherein the contents comprise a signature of the first computer object, behavior information about the first computer object, and information about the first remote computer;
>
> increasing a count associated with a number of times that the first computer object has been seen, and providing the increased count associated with the number of times that the first computer object has been seen to the central server; and
>
> receiving, at a second threat server, at least a portion of the contents of the at least one database associated with the central server, wherein the at least a portion of the contents of the at least one database associated with the central server include a subset of the details of the first computer object stored in the second database.

324. To the extent the preamble is construed as limiting, the Accused Products include a method for *classifying malware* as explained below.

325. The Accused Products perform a method that includes *receiving, at a first threat server details of a first computer object from a first remote computer wherein the details of the first computer object include data uniquely identifying the first computer object*. For example, Sophos' Firewall connects to the Sophos endpoint protection service installed at a remote computer through Sophos Heartbeat. Through that connection to the remote computer, Sophos Firewall receives data about computer objects and, on information and belief, like "all firewalls," uses a

static application signature received from the endpoints to identify computer objects. Cryptographically strong hashes such as MD5 and SHA1, as shown below, uniquely identify an object (*e.g.*, file named "malicious.pdf").



What Next-Gen Firewalls See Today

What Sophos Firewall Sees

You can't control what you can't see. All firewalls today depend on static application signatures to identify apps. But those don't work for most custom, obscure, evasive, or any apps using generic HTTP or HTTPS.

Sophos Firewall utilizes Synchronized Security to automatically identify, classify, and control all unknown applications easily blocking the apps you don't want and prioritizing the ones you do.

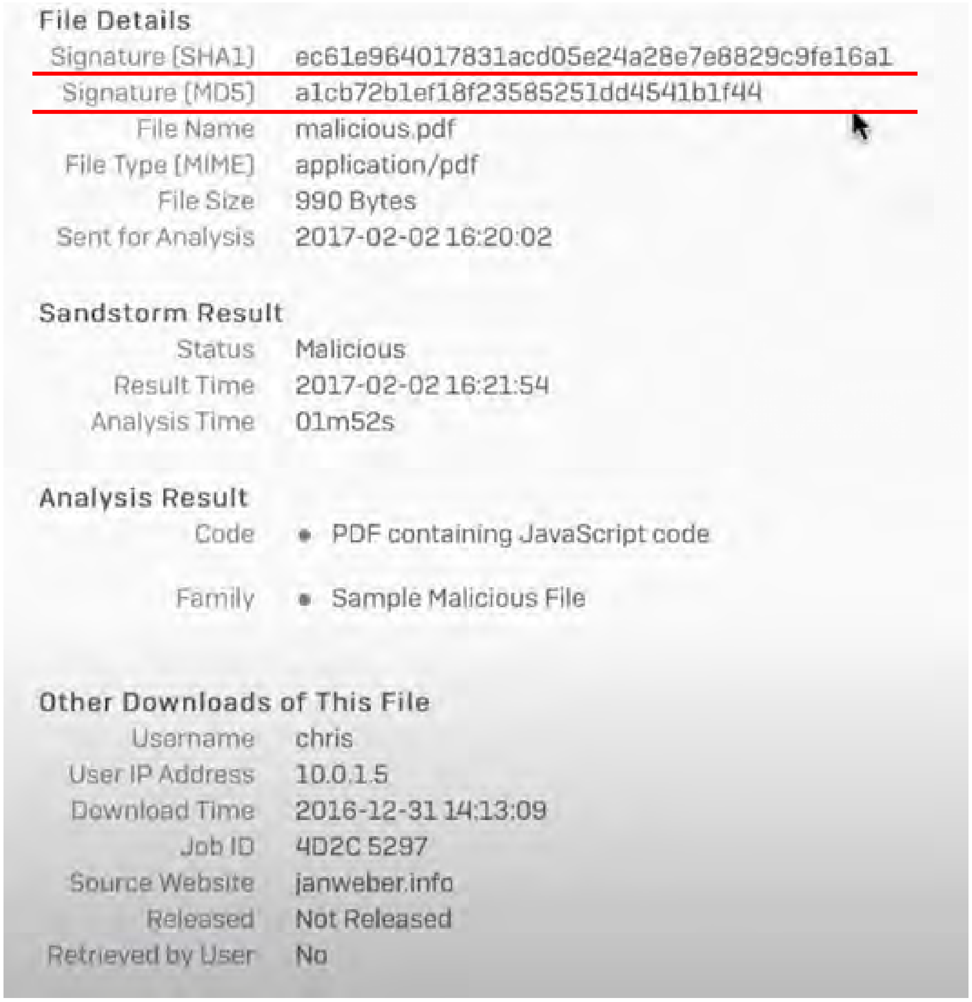
Lateral Movement Protection

Lateral Movement Protection automatically isolates compromised systems at every point in the network to stop attacks dead in their tracks. Healthy endpoints assist by ignoring all traffic from unhealthy endpoints, enabling complete isolation, even on the same network segment, to prevent threats and active adversaries from spreading or stealing data.

Synchronized User ID

User authentication is critically important in a next-generation firewall but often challenging to implement in a seamless and transparent way. Synchronized User ID eliminates the need for client or server authentication agents by sharing user identity between the endpoint and the firewall through Security Heartbeat. It's just another great benefit of having your firewall and endpoints integrated and sharing information.

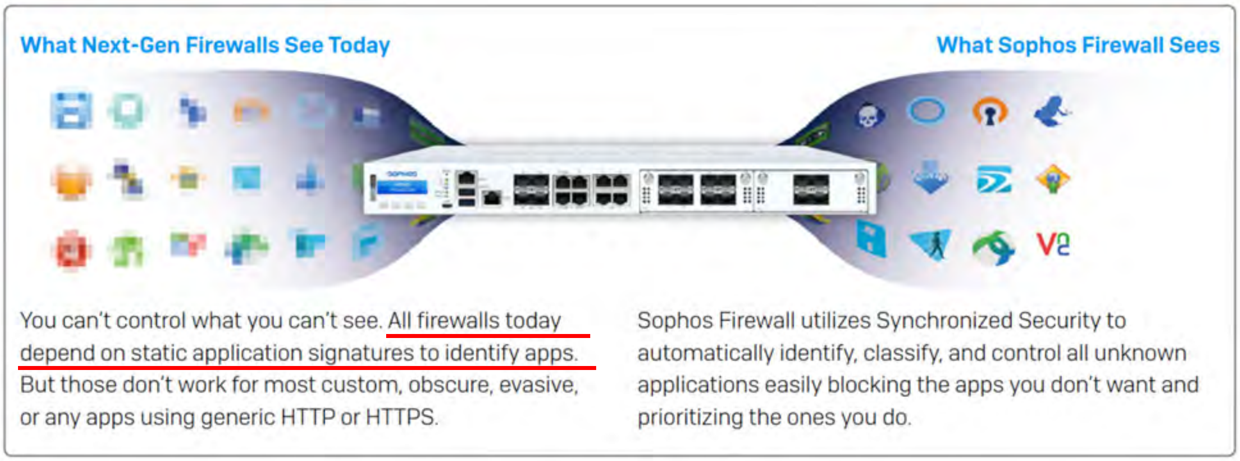
(*See* WBR_SOP000339, https://www.sophos.com/en-us/medialibrary/pdfs/factsheets/sophos-firewall-br.pdf (annotations added).)



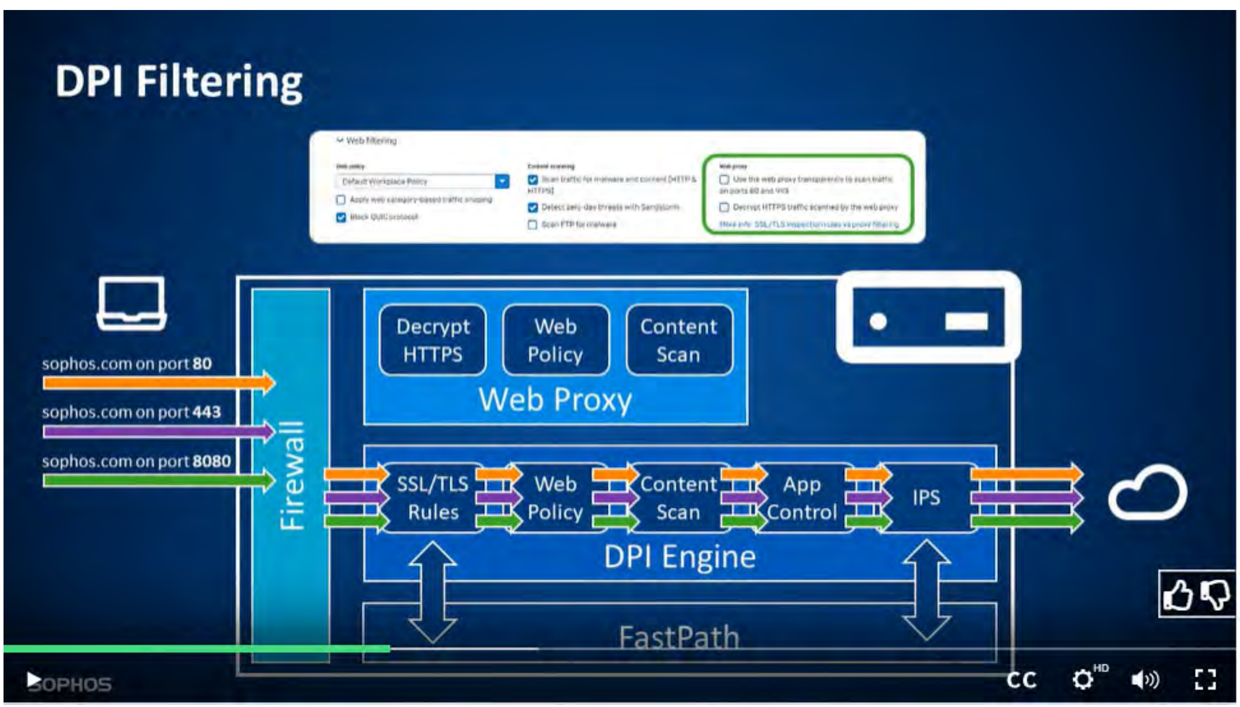
(*See* WBR_SOP000369, https://www.youtube.com/watch?v=YR4CR4Sht3A.)

326. The Accused Products perform a method that includes *determining, by the first threat server whether the first computer object has been previously seen by comparing the data uniquely identifying the first computer object to a plurality of data uniquely identifying plural computer objects in a first database associated with the first threat server*. As explained above, Sophos Firewall receives data about computer objects and, on information and belief, like "all firewalls," uses a static application signature received from the endpoints to identify computer objects. For example, if the Firewall does not have the data in its CPU to enact Fastpath —a mechanism by which trusted data or objects bypass security measures—automatically, Sophos

Firewall looks at static signature data from the unique computer object and compares that information with information provided by Sophos' Talos IPS Signature database. The object then travels along two paths: Sophos Fastpath or a standard protocol. Fastpath is reserved for known permissible objects, and the standard track is used for unknown or malicious objects.



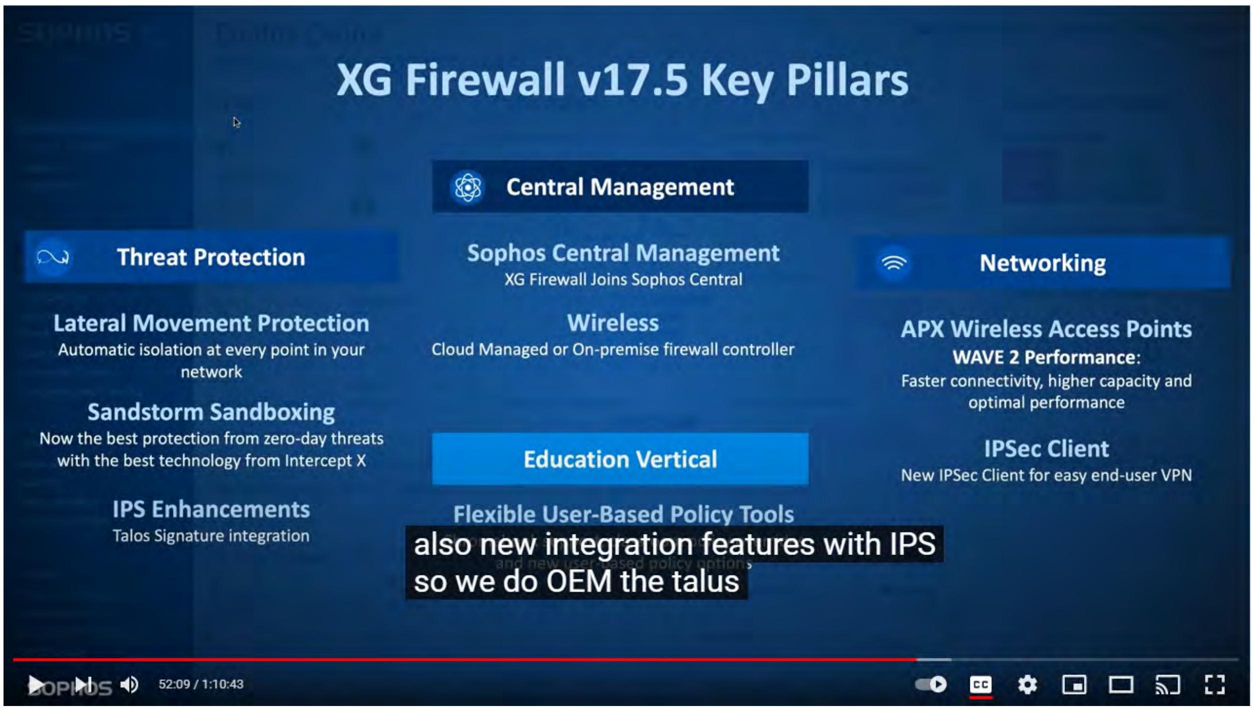
(*See* WBR_SOP000339, https://www.sophos.com/en-us/medialibrary/pdfs/factsheets/sophos-firewall-br.pdf.)



(*See* WBR_SOP000491, https://techvids.sophos.com/watch/uCC7QqkYcTJtLiBMNhZV32.)

FastPath network flow offloads (bypasses processing of) trusted traffic. Offloading eliminates the need to apply full firewall processing to every packet in a connection, minimizing the use of processing cycles.

(*See* WBR_SOP000496, https://docs.sophos.com/nsg/sophos-firewall/18.0/Help/en-us/webhelp/onlinehelp/nsg/sfos/concepts/Architecture.html; *see also* WBR_SOP000369, https://www.youtube.com/watch?v=YR4CR4Sht3A.)



(*See* https://www.youtube.com/watch?v=aXNo4V2A1Gw.)

> **MTR/XDR Ready**
> Sophos MTR provides optional 24/7 threat hunting, detection and response delivered by an expert team as a fully-managed service. Sophos XDR offers extended detection and response managed by your own team. Regardless of whether you manage it yourself, or Sophos manages it for you, your Sophos Firewall is ready to share the necessary threat intelligence and data to the cloud.
> Central Orchestration is included in the Xstream Protection bundle and is available for separate purchase.
> * Expected soon.

(*See* WBR_SOP000339, https://www.sophos.com/en-us/medialibrary/pdfs/factsheets/sophos-firewall-br.pdf.)

327. The Accused Products perform a method that includes *receiving additional information about the first computer object from the first remote computer when the first computer object has not been previously seen*. For example, if Sophos Firewall cannot immediately identify the object based on its Fastpath data or there is no match in Talos database, then Sophos Firewall receives more information about the unknown object through Sophos Heartbeat. As explained below, that additional data is provided to an Advanced Threat Protection ("ATP") database, which "provides" information on "JavaScript emulation, behavioral analysis, and origin reputation" based on the information received from the endpoints.

Synchronized Security

Security Heartbeat™: Your firewall and your endpoints are finally talking

Sophos Firewall is the only network security solution that is able to fully identify the user and source of an infection on your network, and automatically limit access to other network resources in response. This is made possible with our unique Sophos Security Heartbeat that shares telemetry and health status between Sophos endpoints and your firewall and integrates endpoint health into firewall rules to control access and isolate compromised systems.

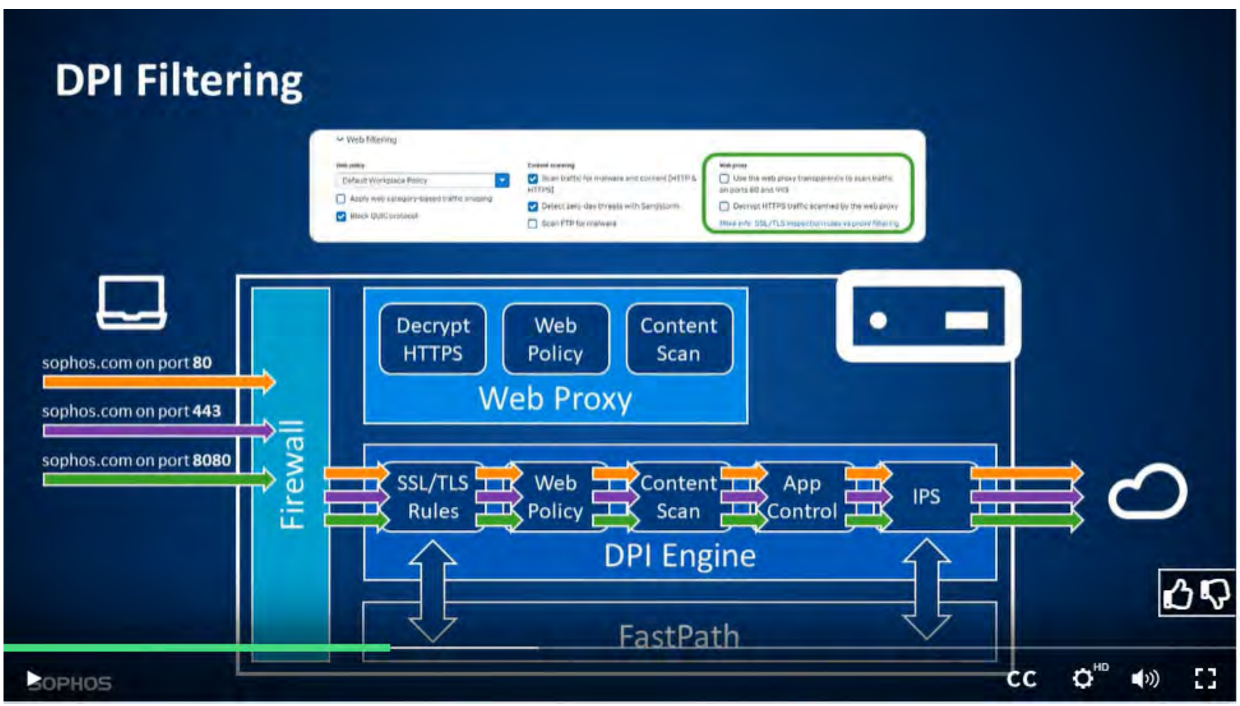
The good news is, this all happens automatically, and is successfully helping numerous businesses and organizations to save time and money in protecting their environments today.

Synchronized Application Control

Using Security Heartbeat, we can do much more than just see the health status of an endpoint. We also have a solution to one of the biggest problems most network administrators face today - lack of visibility into network traffic.

Synchronized Application Control utilizes the Heartbeat connections with Sophos endpoints to automatically identify, classify, and control application traffic. All encrypted, custom, evasive, and generic HTTP or HTTPS applications which are currently going unidentified will be revealed.

(*See* WBR_SOP000339, https://www.sophos.com/en-us/medialibrary/pdfs/factsheets/sophos-firewall-br.pdf.)



(*See* WBR_SOP000491, https://techvids.sophos.com/watch/uCC7QqkYcTJtLiBMNhZV32.)

Network Protection

- **Xstream TLS Inspection:** TLS 1.3 inspection with pre-packaged exceptions
- **Xstream DPI engine:** streaming deep-packet inspection
- **IPS:** Next-gen Intrusion Prevention
- **ATP:** Advanced Threat Protection
- **Synchronized Security Heartbeat:** integration with Sophos Endpoints to identify and isolate threats
- **Clientless VPN:** HTML5
- **SD-RED VPN:** Manage SD-RED devices
- **Reporting:** Extensive network and threat reporting

Web Protection

- **Xstream TLS Inspection:** TLS 1.3 inspection with pre-packaged exceptions
- **Xstream DPI engine:** streaming deep-packet inspection
- **Web Control:** by user, group, category, URL, keyword
- **Web Threat Protection:** malware, PUA, malicious JavaScript, Pharming
- **App Control:** by user, group, category, risk, and more
- **Synchronized App Control:** integration with Sophos endpoints to identify unknown apps
- **Synchronized SD-WAN:** utilizing Synchronized App Control to route unknown apps
- **Reporting:** Extensive web and app reporting

Zero-Day Protection

- **Xstream TLS Inspection:** TLS 1.3 inspection with pre-packaged exceptions
- **Xstream DPI engine:** streaming deep-packet inspection
- **Zero-Day Threat Protection:** analyze all unknown files using AI, ML, and sandboxing
- **Powered by SophosLabs Intelix:** cloud-based intelligence and analysis
- **Machine Learning:** using multiple deep learning models
- **Cloud Sandboxing:** dynamic run-time analysis of unknown files
- **Reporting:** Extensive threat intelligence analysis reporting

(*See* WBR_SOP000339, https://www.sophos.com/en-us/medialibrary/pdfs/factsheets/sophos-firewall-br.pdf.)

Advanced Web Threat Protection

Backed by SophosLabs, our advanced engine provides the ultimate protection from today's polymorphic and obfuscated web threats. Innovative techniques like JavaScript emulation, behavioral analysis, and origin reputation help keep your network safe.

(*See* WBR_SOP000339, https://www.sophos.com/en-us/medialibrary/pdfs/factsheets/sophos-firewall-br.pdf.)

328. The Accused Products perform a method that includes *storing the details of the first computer object and the received additional information about the first computer object in a*

second database associated with the first threat server when the first computer object has not been previously seen. For example, after Sophos Firewall determines that it has not previously seen the computer object, it then sends the information it has received about the object to the Advanced Threat Protection ("ATP") database associated with that Sophos Firewall. The ATP database collects and "provides" information on "JavaScript emulation, behavioral analysis, and origin reputation." On information and belief, each Sophos Firewall is associated with an ATP database. For example, after a first firewall encounters a new object, the system administrator must update other firewalls on the network with the new information from the first firewall, thereby indicating there are multiple ATP databases.

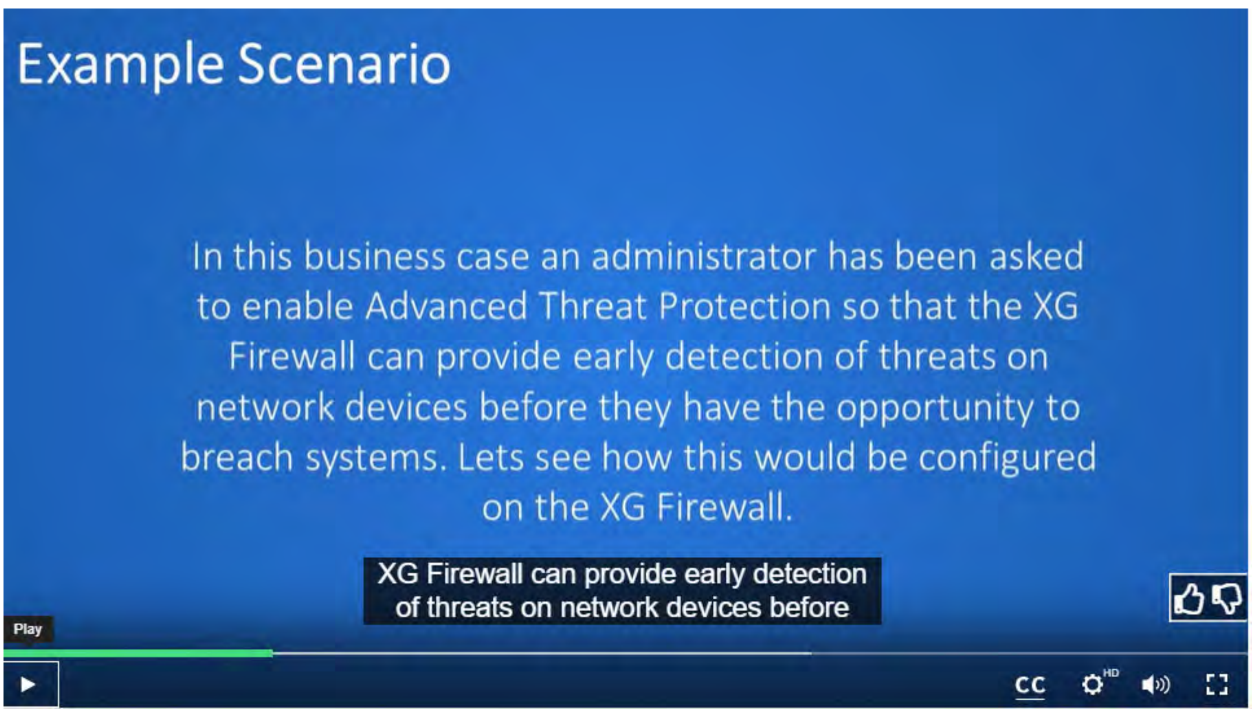
> **Advanced Web Threat Protection**
> Backed by SophosLabs, our advanced engine provides the ultimate protection from today's polymorphic and obfuscated web threats. Innovative techniques like JavaScript emulation, behavioral analysis, and origin reputation help keep your network safe.

(*See* WBR_SOP000339, https://www.sophos.com/en-us/medialibrary/pdfs/factsheets/sophos-firewall-br.pdf.)

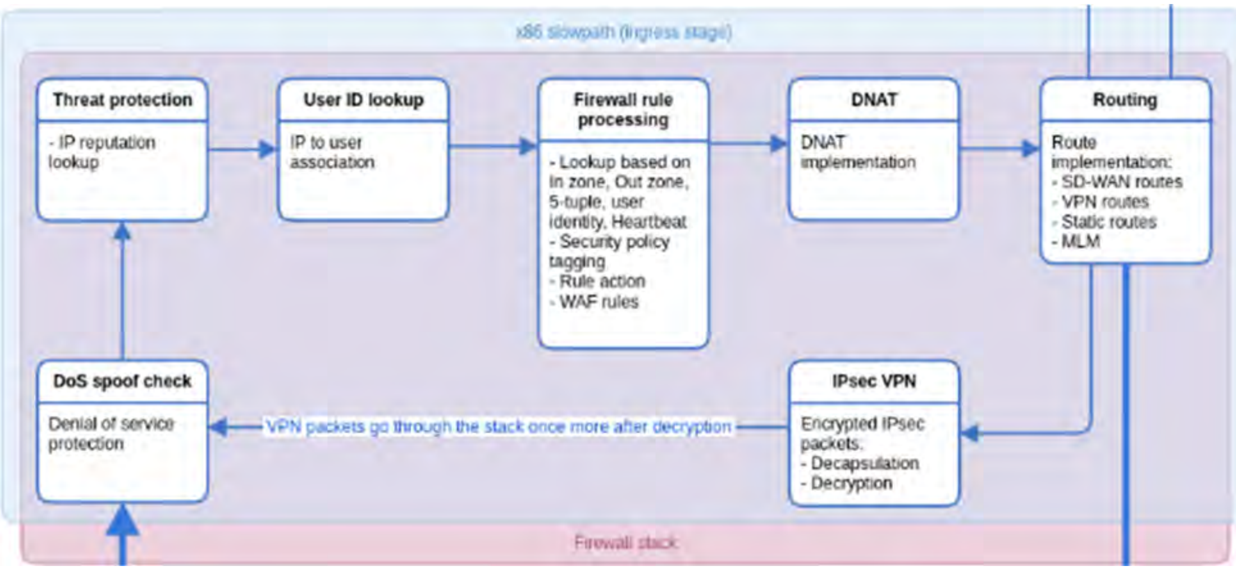
Powered by the industry-leading SophosLabs, the Zero-Day Protection subscription includes a fully cloud-based threat intelligence and threat analysis platform. This provides deep learning-based file analysis, detailed analysis reporting, and a threat meter to show the risk summary for a file.

We use layers of analytics to identify known and potential threats, reduce unknowns, and derive verdicts and intelligence reports for the most commonly used file types.

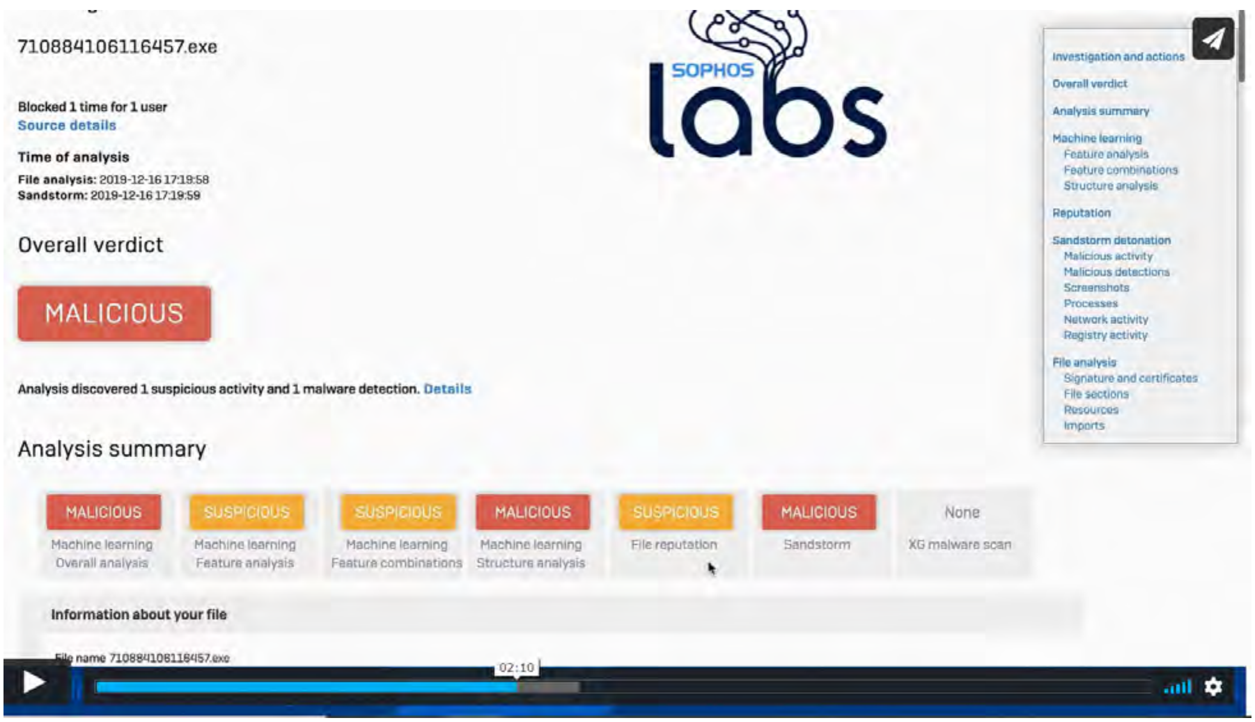
(*See* WBR_SOP000339, https://www.sophos.com/en-us/medialibrary/pdfs/factsheets/sophos-firewall-br.pdf.)



(*See* WBR_SOP000487, https://techvids.sophos.com/watch/wm84yg3wcZtYB1sZjwYyKt.)

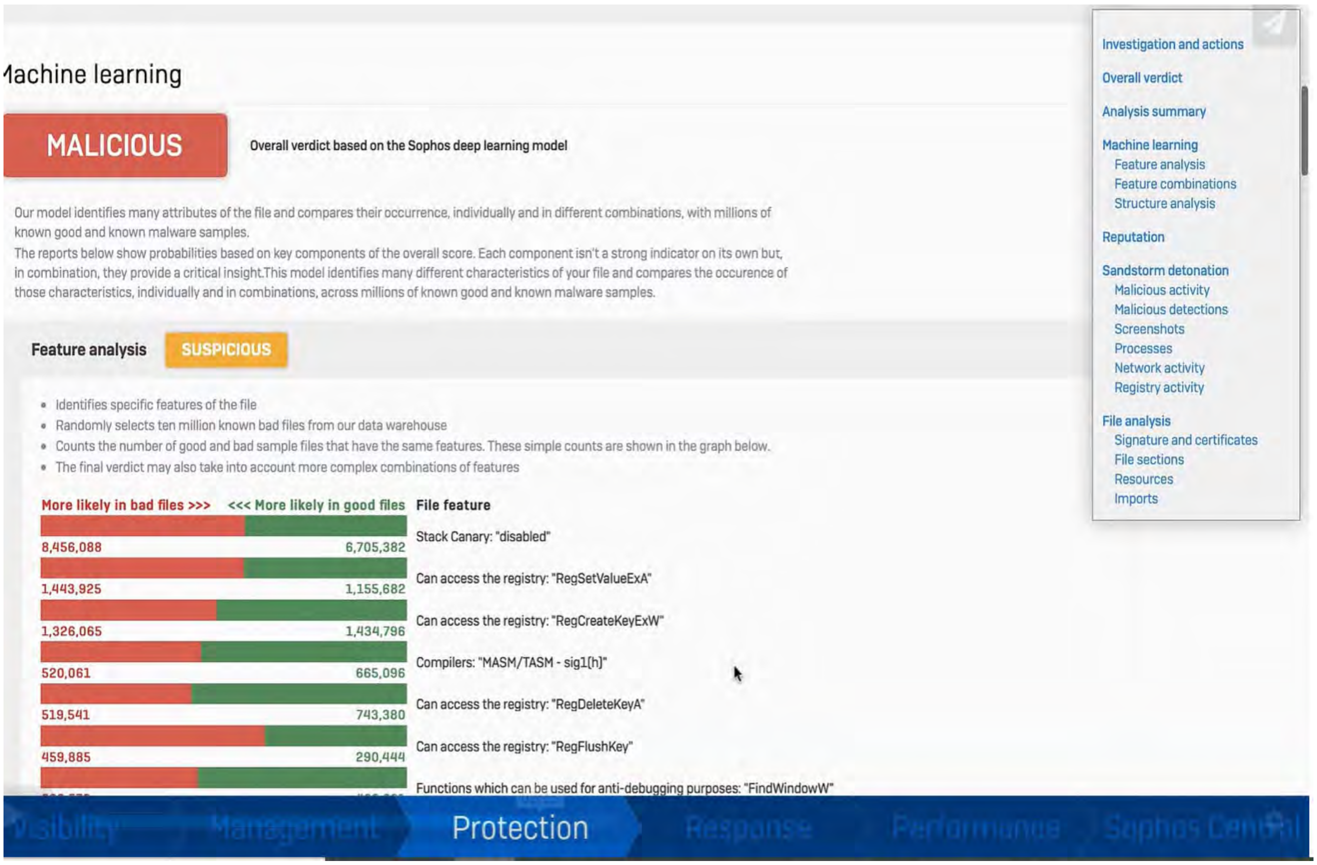


(*See* WBR_SOP000531, https://community.sophos.com/sophos-xg-firewall/f/recommended-reads/122357/life-of-a-packet-sophos-firewall#mcetoc_1fc8lebu84.)



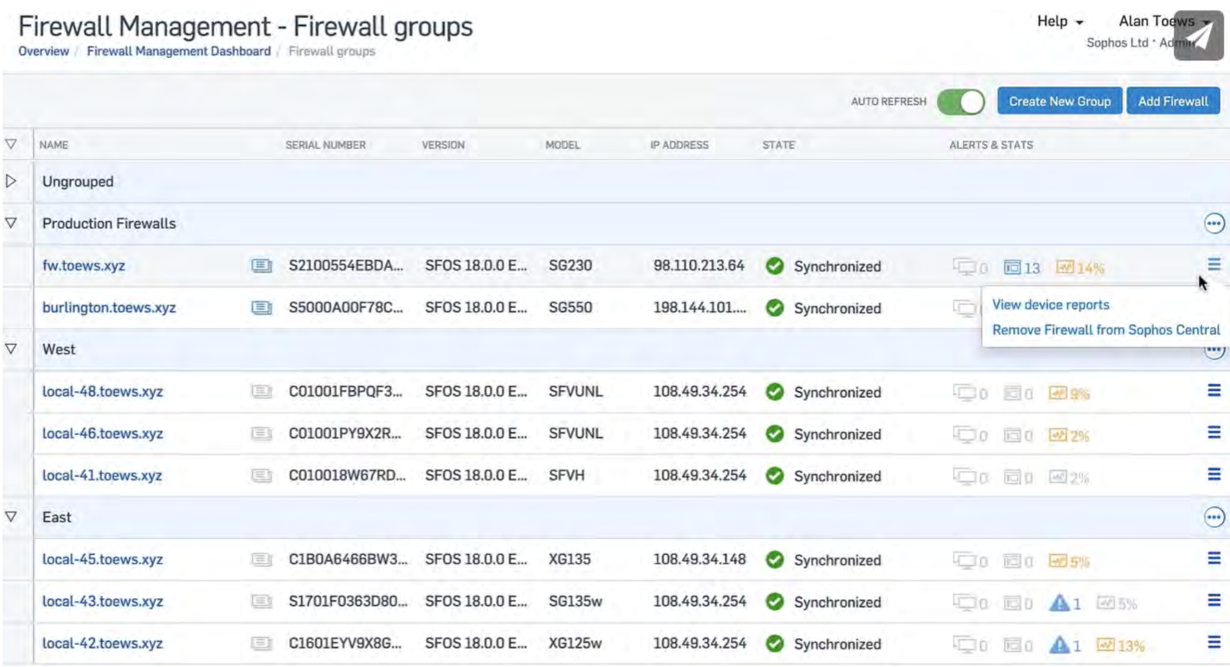
(*See* WBR_SOP000489,

https://player.vimeo.com/video/144094496?width=800&height=450&iframe=true& portrait=0.)



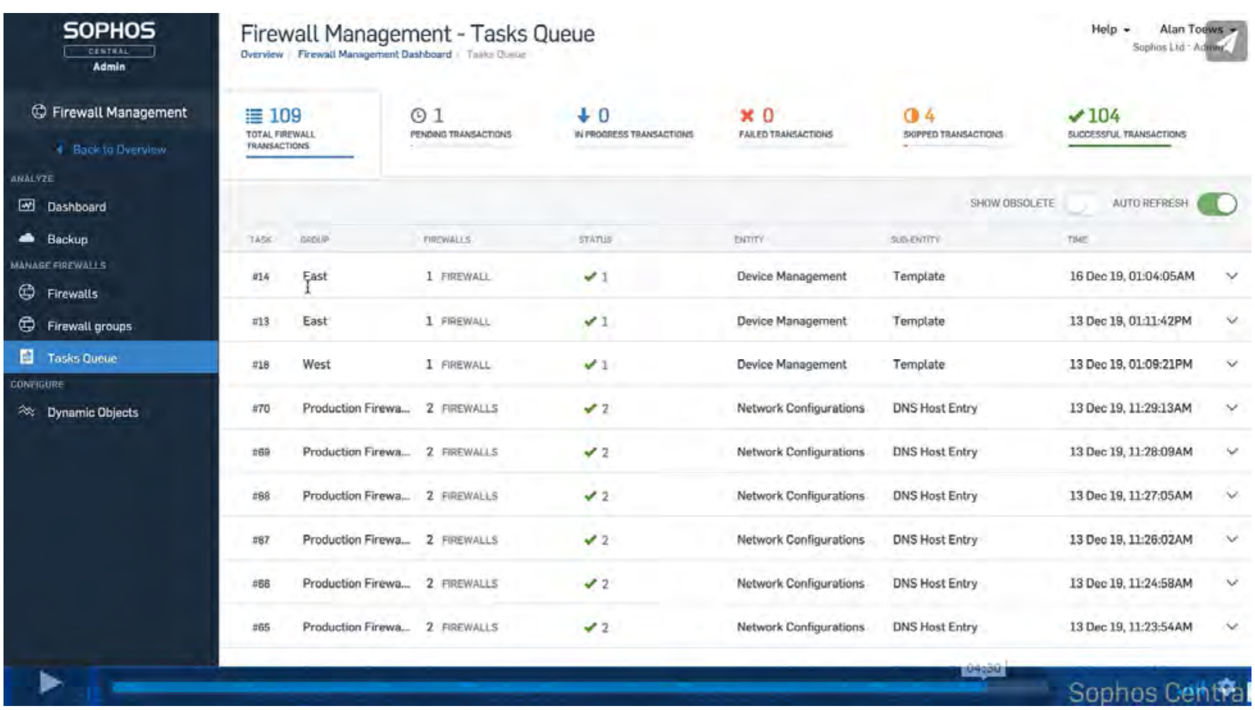
(*See* WBR_SOP000489,

https://player.vimeo.com/video/144094496?width=800&height=450&iframe= true&portrait=0.)



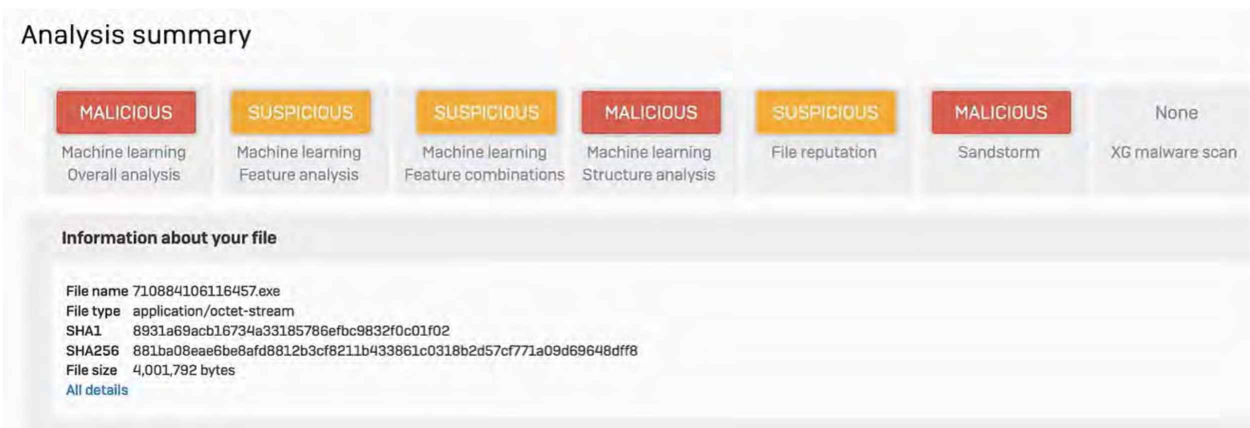
(*See* WBR_SOP000489,

https://player.vimeo.com/video/144094496?width=800&height=450&iframe= true&portrait=0.)

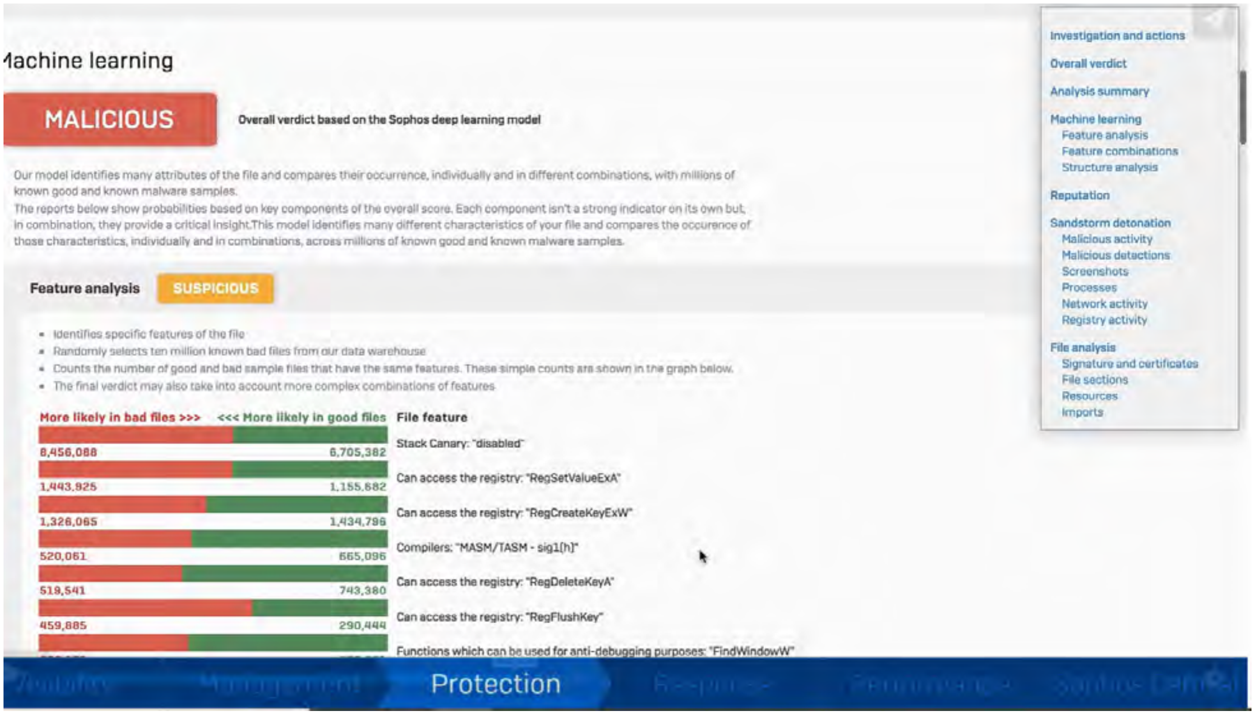


(*See* WBR_SOP000489, https://player.vimeo.com/video/144094496?width=800&height=450&iframe= true&portrait=0.)

329. The Accused Products perform a method that includes *providing contents of the second database to at least one database associated with a central server, wherein the contents comprise a signature of the first computer object, behavior information about the first computer object, and information about the first remote computer*. For example, on information and belief, the ATP database associated with the Firewall that receives the data from the first remote computer communicates with a reporter database associated with Sophos' Control Center, Sophos Central. Based on the information collected and reported, Sophos Central displays information about the object's signature, behavior, and the computer on which it was observed.

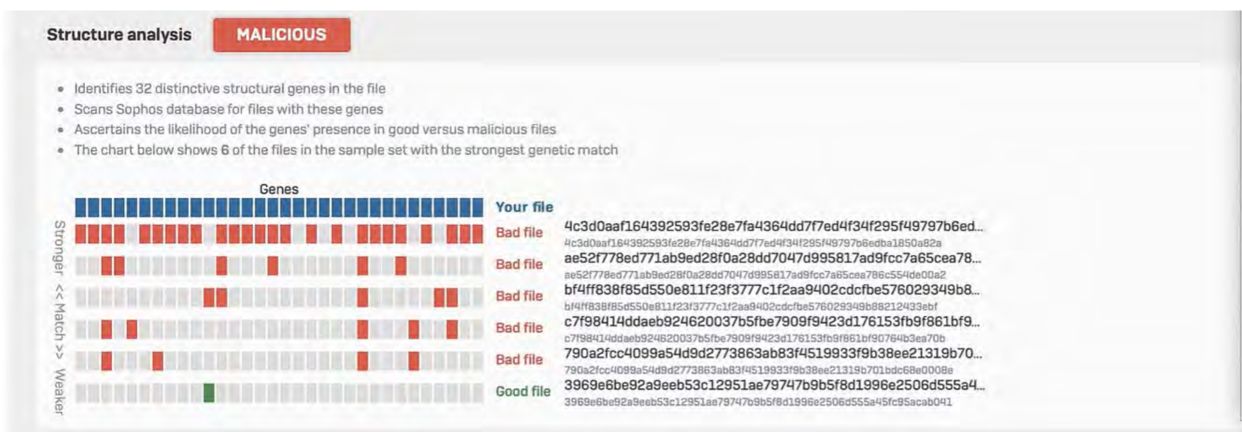


(*See* WBR_SOP000489, https://player.vimeo.com/video/144094496?width=800&height=450&iframe= true&portrait=0.)



(*See* WBR_SOP000489,

https://player.vimeo.com/video/144094496?width=800&height=450&iframe= true&portrait=0.)

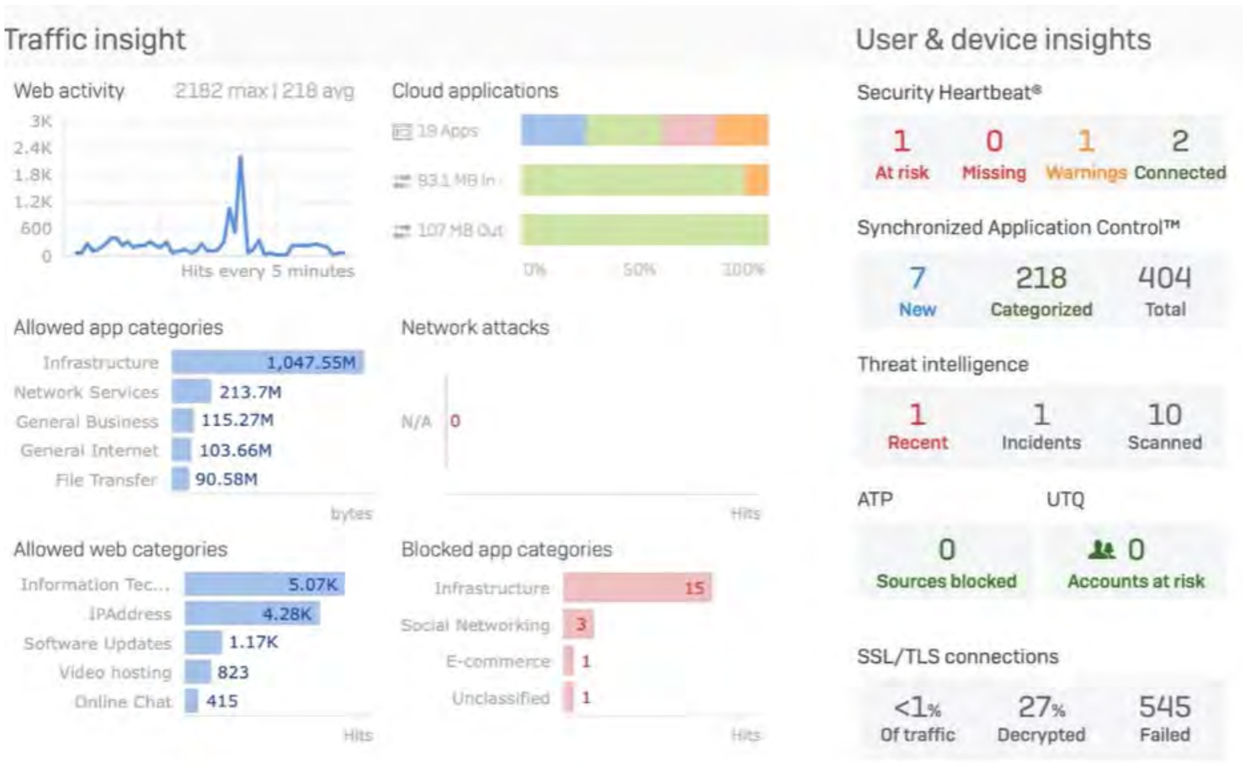


(*See* WBR_SOP000489,

https://player.vimeo.com/video/144094496?width=800&height=450&iframe= true&portrait=0.)

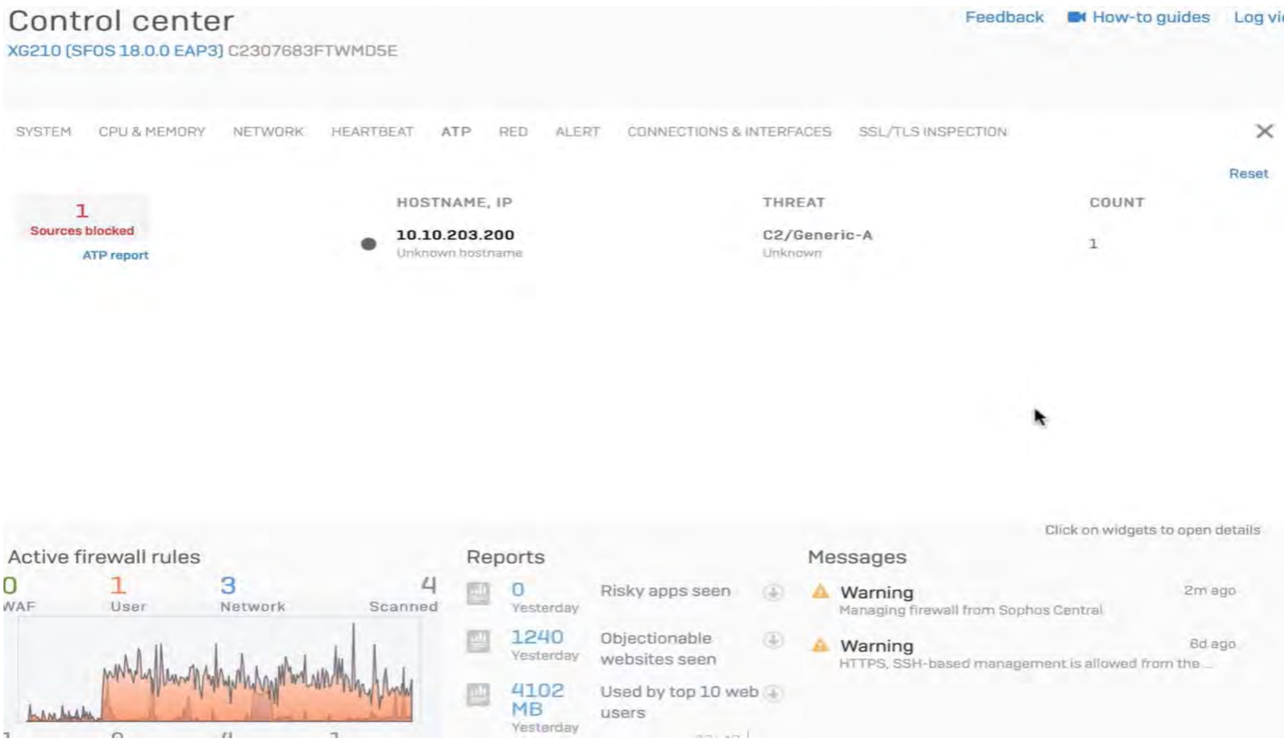
330. The Accused Products perform a method that includes *increasing a count associated with a number of times that the first computer object has been seen and providing the increased count associated with the number of times that the first computer object has been seen*

to the central server. For example, Sophos Central displays a count for the number of times the object under investigation has been seen. It then updates the count based on the current observation.



(*See* WBR_SOP000489,

https://player.vimeo.com/video/144094496?width=800&height=450&iframe= true&portrait=0.)

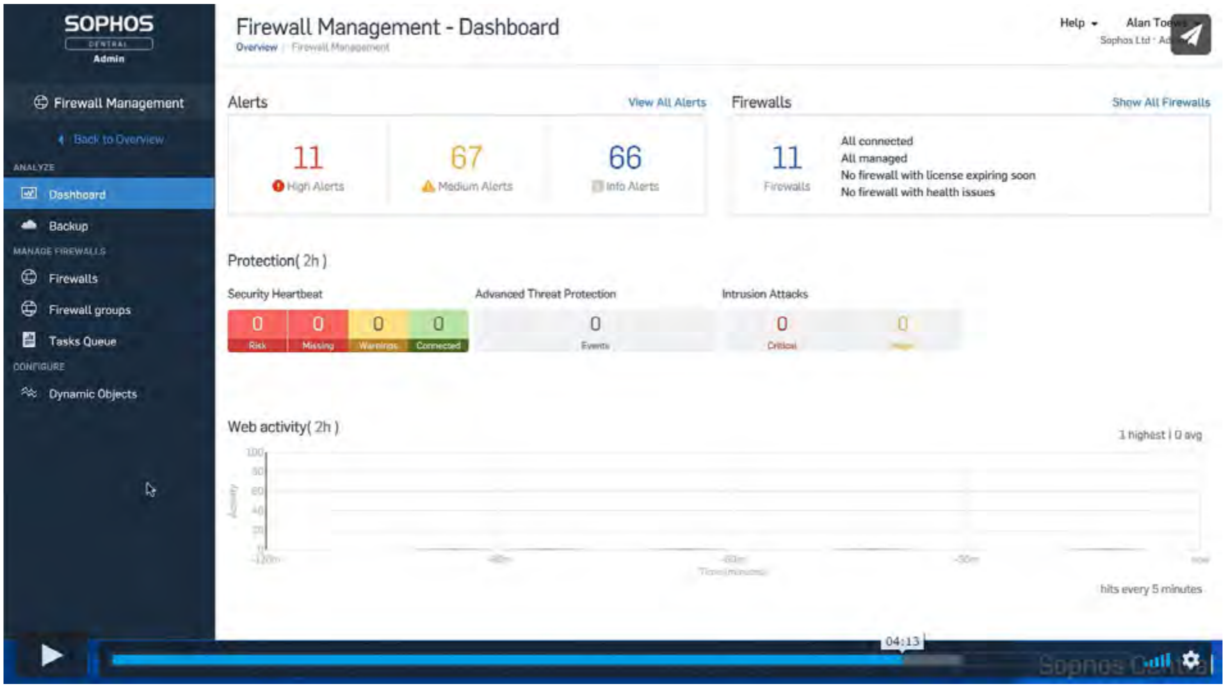


(*See* WBR_SOP000489,

https://player.vimeo.com/video/144094496?width=800&height=450&iframe= true&portrait=0.)

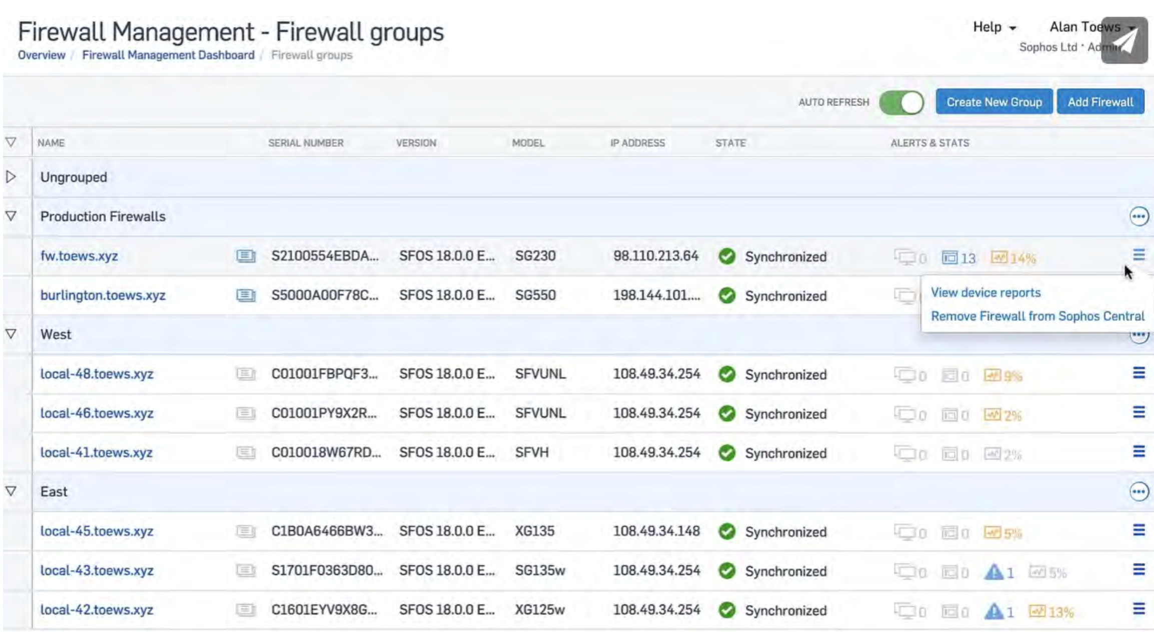
331. The Accused Products perform a method that includes *receiving, at a second threat server at least a portion of the contents of the at least one database associated with the central server wherein the at least a portion of the contents of the at least one database associated with the central server include a subset of the details of the first computer object stored in the second database*. For example, based on the information received at Sophos Central from the first Sophos Firewall, Sophos Central sends at least some of the information about the computer object to each of the other ATP databases associated with each of the Sophos Firewalls (*i.e.*, other Sophos Firewalls connected to the network)—such as information that enables the object to be identified (*e.g.*, on information and belief, information that enables each Sophos Firewall to use a static

application signature to identify computer objects). The updating of the ATP databases is demonstrated by the update to the Firewall rules to look for the new object.



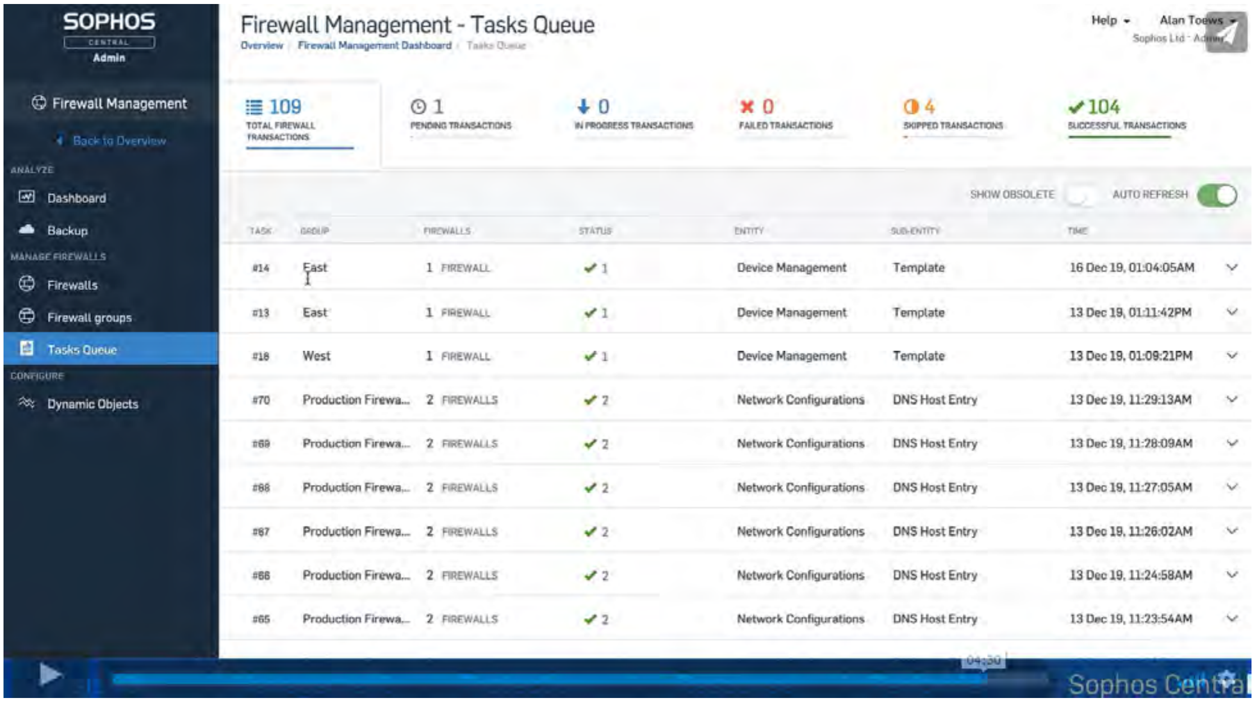
(*See* WBR_SOP000489,

https://player.vimeo.com/video/144094496?width=800&height=450&iframe= true&portrait=0.)



(*See* WBR_SOP000489,

https://player.vimeo.com/video/144094496?width=800&height=450&iframe= true&portrait=0.)



(*See* WBR_SOP000489,

https://player.vimeo.com/video/144094496?width=800&height=450&iframe= true&portrait=0.)

332. Each claim in the ’721 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ’721 Patent.

333. Defendant has been aware of the ’721 Patent since at least the filing of this Complaint. Further, Plaintiffs have marked its products with the ’721 Patent, including on its web site, since at least July 2020.

334. Defendant directly infringes at least claim 1 of the ’721 Patent, literally or under the doctrine of equivalents, by performing the steps described above. For example, on information and belief, Defendant performs the claimed method in an infringing manner as described above by running this software and corresponding systems to protect its own computer and network operations. On information and belief, Defendant also performs the claimed method as described

above when testing the operation of the Accused Products and corresponding systems. As another example, Defendant perform each of the method steps as described above when providing or administering services to third parties, customers, and partners using the Accused Products.

335. Defendant's partners, customers, and end users of the Accused Products and corresponding systems and services directly infringe at least claim 1 of the '721 Patent, literally or under the doctrine of equivalents, at least by using the Accused Products and corresponding systems and services, as described above.

336. Defendant has actively induced and is actively inducing infringement of at least claim 1 of the '721 Patent with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b). For example, Defendant encourages and induces customers with specific intent to use Sophos security software in a manner that infringes claim 1 of the '721 Patent at least by offering and providing software that performs a method that infringes claim 1 when installed and operated by the customer, and by engaging in activities relating to selling, marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including the activities described below.

337. Defendant encourages, instructs, directs, and/or requires third parties—including its certified partners and/or customers—to perform the claimed method using the software, services, and systems in infringing ways, as described above.

338. Defendant further encourages and induces its customers to infringe claim 1 of the '721 Patent: 1) by making its security services available on its website, providing applications that allow users to access those services, widely advertising those services, and providing technical support and instructions to users, and 2) through activities relating to marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including Sophos

security software, and services in the United States. (*See* https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx; WBR_SOP000525, https://partners.sophos.com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1; WBR_SOP000209, https://secure2.sophos.com/en-us/security-news-trends/whitepapers/gated-wp/cybersecurity-system-buyers-guide.aspx%23formFrame; *see also* WBR_SOP000214, https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophos-synchronized-security-ds.pdf.)

339. For example, on information and belief, Defendant shares instructions, guides, and manuals, which advertise and instruct third parties on how to use the software as described above, including to at least customers and partners. (*Id*.) On further information and belief, Defendant also provides customer service and technical support to purchasers of the Accused Products and corresponding system and services, which directs and encourages customers to perform certain actions that use the Accused Products in an infringing manner. (*Id*.)

340. Defendant and/or its partners recommend and sell the Accused Products and provide technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Defendant and/or one of its partners, which obligates each customer to perform certain actions in order to use the Accused Products. Further, in order to receive the benefit of Defendant's or its partner's continued technical support and their specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that infringes the '721 Patent.

341. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Product remains operational for each

customer through ongoing technical support, on information and belief, Defendant and/or its partners affirmatively aid and abet each customer's use of the Accused Products in a manner that performs the claimed method of, and infringes, the '721 Patent.

342. Defendant also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the claimed methods, and not a staple article or commodity of commerce suitable for substantial non-infringing uses. Indeed, as shown above, the Accused Products and the example functionality described below have no substantial non-infringing uses but are specifically designed to practice the '721 Patent.

343. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the Sophos database and server architecture and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. For example, without compiling and correlating threat information about an assessed object from dynamically updated threat databases, the Accused Products could not assess whether to allow those objects to pass through firewalls or subject them to further analysis, triage and final designation as malicious or benign. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the '721 Patent, that functionality could not be performed.

344. Additionally, the accused functionality, including the Sophos database and server architecture and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without leveraging information from multiple dynamically updated threat databases, the Accused Products could not propagate the latest intelligence about objects to their firewalls, and therefore could not enable their firewalls to make accurate decisions about the malicious or benign nature of assessed objects. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the '721 Patent, that functionality could not be performed.

345. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Defendant constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionalities (including Sophos database and server architecture) constitute a material part of the inventions claimed at least because such functionality is integral to the processes identified above (such as "receiving, at a first threat server, details of a first computer object…"; "determining, by the first threat server, whether the first computer object has been previously seen…"; "receiving additional information about the first computer object…"; "storing the details of the first computer object…"; "providing contents of the second database…"; "increasing a count associated with a number of times that the first computer object has been seen…"; "receiving, at a second threat server, at least a portion of

the contents of the at least one database associated with the central server….”), as recited in the claims of the ’721 Patent. None of these products are staple goods—they are sophisticated and customized cyber security and malware detection products, methods, and systems.

346. Defendant’s infringing actions have continued after the original Complaint was filed. Defendant had knowledge of the ’721 Patent and of the specific acts that constitute infringement of the ’721 Patent at least based on the original Complaint, but has continued to engage in its infringing activity, including by selling, making, configuring, and installing the Accused Products and performing the accused functionality, and by engaging in activities that constitute inducing infringement and contributory infringement as described above.

347. On information and belief, the infringing actions of each partner, customer, and/or end-user of the Accused Products are attributable to Defendant. For example, on information and belief, Defendant directs and controls the activities or actions of its partners or others in connection with the Accused Products by contractual agreement or otherwise requiring partners or others to provide information and instructions to customers who acquire the Accused Products which, when followed, results in infringement. Defendant further directs and controls the operation of devices executing the Accused Products by programming the software which, when executed by a customer or end user, perform the method steps of at least claim 1 of the ‘721 Patent.

348. Plaintiffs have suffered and continue to suffer damages, including lost profits, as a result of Defendant’s infringement of the ’721 Patent. Defendant is therefore liable to Plaintiffs under 35 U.S.C. § 284 for damages in an amount that adequately compensates Plaintiffs for Defendant’s infringement, but no less than a reasonable royalty.

349. Plaintiffs will continue to suffer irreparable harm unless this Court preliminarily and permanently enjoins Defendant, its agents, employees, representatives, and all others acting

in concert with Defendant from infringing the ’721 Patent.

350. Defendant’s infringement of the ’721 Patent is knowing and willful. Defendant had actual knowledge of the ’721 Patent at least by the time Plaintiffs filed this lawsuit and had constructive knowledge of the ’721 Patent from at least the date Plaintiffs marked their products with the ’721 Patent and/or provided notice of the ’721 Patent on their website.

351. On information and belief, despite Defendant’s knowledge of the Asserted Patents and Plaintiffs’ patented technology, Defendant made the deliberate decision to sell products and services that they knew infringe the patents. Defendant’s continued infringement of the ’721 Patent with knowledge of the ’721 Patent constitutes willful infringement.

352. Plaintiffs’ allegations of infringement, indirect infringement, and willful infringement with respect to this patent are further set forth in Plaintiffs’ Disclosure of Asserted Claims and Preliminary Infringement Contentions Pursuant to the Court’s Standing Order Governing Patent Proceedings served July 12, 2022.

EIGHTH CAUSE OF ACTION
(INFRINGEMENT OF THE ’869 PATENT)

353. Plaintiffs reallege and incorporate by reference the allegations of the preceding paragraphs of this Complaint.

354. Defendant has infringed and continues to infringe one or more claims of the ‘869 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including features such as Intercept X Advanced with EDR (“Intercept X”), at least when used for their ordinary and customary purposes, practice each element of at least claim 46 of the ’869 Patent, as described below.

355. Claim 46 of the ’869 Patent recites:

46. A system comprising:

at least one memory;

at least one processor configured to perform operation of:

identifying static data points that may be indicative of either a harmful or benign executable file; and

associating the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files; and

at least one processor configured to perform operation of:

identifying an executable file to be evaluated;

extracting a plurality of static data points from the executable file;

generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files,

wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range; and

evaluating the feature vector using a machine learning model to determine whether the executable file is harmful.

356. The Accused Products constitute a system comprising each element of claim 46 of the ’869 Patent. To the extent the preamble is construed to be limiting, the Accused Products constitute *a system* of malware detection using machine learning, as further explained below.

357. The Accused Products constitute a system comprising *at least one memory and at least one processor configured to perform operation of identifying static data points that may be indicative of either a harmful or benign executable file and associating the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files*. For example, Intercept X employs “Deep Learning models” (*i.e.*,

advanced machine learning models) that have been trained on "hundreds of millions of samples" to classify features extracted from a file to "determine if a file is benign or malicious…before the file executes." Such samples include "Potentially Unwanted Applications" such as adware. Before using a deep learning model to classify the features extracted from a file, Intercept X creates a "vector of information" that translates those features into data that the model can "intake" and "process."

> **Performance:** Sophos' Deep Learning technology is incredibly fast. In less than 20 milliseconds the model is able to extract millions of features from a file, conduct a deep analysis, and determine if a file is benign or malicious. This entire process happens before the file executes.
>
> **SophosLabs:** One of the most important aspects to any model is the data that used for training. Our team of data scientists are part of the SophosLabs group, granting them access to hundreds of millions of samples. This allows them to create the best possible predictions in our models. The integration between the two groups also leads to better data labeling [and therefore better modeling]. The bi-directional sharing of threat intelligence and real-world feedback between the team of data scientists and threat researchers continuously improves the accuracy of our models.

(*See* WBR_SOP000300, https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophos-intercept-x-deep-learning-dsna.pdf.)

Feature Engineering in Machine Learning

Before creating a machine learning model, it's important to prepare our data. Preparing the data requires translating it into a language our model can understand. This is referred to as feature engineering.

Artificial neural network models intake data as a vector of information, so simply feeding the model a URL – which is not in the language of a vector – means that the model can't process it without some manipulation. There are countless ways that samples can be translated into features, though it takes some domain knowledge to do so. Using the URL example again, one way to translate a URL into a usable language is through a combination of ngramming and hashing. Ngrams are a popular method in DNA sequencing research. For example, the results of a three-gram ngram for the URL "https://sophos.com/company/careers.aspx" would be:

['htt', 'ttp', 'tps', 'ps:', 's:/', '://', '//s', '/so', 'sop', 'oph', 'pho', 'hos', 'os.', 's.c', '.co', 'com', 'om/', 'm/c', '/co', 'com', 'omp', 'mpa', 'pan', 'any', 'ny/', 'y/c', '/ca', 'car', 'are', 'ree', 'eer', 'ers', 'rs.', 's.a', '.as', 'asp', 'spx']

Once the ngrams are calculated, we need to translate them into a numerical representation. This can be done through a hashing mechanism. We will create an n-length long vector – say 1000 – and hash each ngram using a hashing algorithm. The resulting number from the hash of a particular ngram will be the index of which we will add 1. For example, if the first ngram 'htt' results in a hash of three and our vector is five units long, the result would be [0, 0, 1, 0, 0]. We continue this process for every ngram and for every URL until we have the list of URLs completely transformed into individual n-length vectors. When using this method for our toy model, these vectors will be 1,000 units long.

(*See* WBR_SOP000302, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/machine-learning-how-to-build-a-better-threat-detection-model.pdf?cmp=70130000001xKqzAAE.)

2.1 MLP Model

An MLP is a class of feedforward ANN with an input layer, an output layer, and one or more hidden layers between them. Each node in one layer fully connects to every node in the following layer. Except for nodes in the input layer, each node is a neuron (or processing element) with a nonlinear activation function associated with a scalar weight which is adjusted during training. An MLP is a supervised learning algorithm that learns to identify data patterns for classification or regression. In order to carry out the learning process, one needs to extract a digital representation $\boldsymbol{x}$ of a given object or event that needs to be fed into the MLP. The learning task becomes to find a multidimensional function $\Phi(\cdot)$ between input $\boldsymbol{x}$ and target $\boldsymbol{y}$

$$\boldsymbol{y} \cong \Phi(\boldsymbol{x}) \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^N$, a real-valued input feature vector $\boldsymbol{x} = [x_1, \dots, x_N]^T$ in an N dimensional feature space, with $(\cdot)^T$ denoting the transpose operation. Similarly, $\boldsymbol{y} \in \mathbb{R}^M$, a real-valued target classification vector $\boldsymbol{y} = [y_1, \dots, y_M]^T$ in an M dimensional classification space. In other words, an MLP learning process is to find $\Phi(\cdot)$

(*See* WBR_SOP000318, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophos-black-hat-2018-technicalpaper.pdf.)

Generic ML PUA detections

Potentially Unwanted Application (PUA) is a term used to describe applications that, while not malicious, are generally considered unsuitable for business networks.

A Generic ML PUA detection is generated by Sophos Intercept X's Machine Learning (ML) engine, also referred to by the specific Sophos approach Deep Learning and is designed to detect PUAs in PE (Portable Executable) files such as:

- .exe
- .sys
- .dll
- .pif
- .scr
- and many more

If a detection of this type has been received, it is because Sophos has detected a file on the computer that our Deep Learning threat model has decided is a PUA. This is a pre-execution detection meaning the file was detected before it was able to be run.

Major PUA classifications

The major PUA classifications are:

- adware
- dialer
- non-malicious spyware
- remote administration tools
- hacking tools

(*See* WBR_SOP000318, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophos-black-hat-2018-technicalpaper.pdf.)

358. These features include the most "significant strings found in the file," the attribute "Findcrypt," which "shows any suspicious cryptographic constants" found within the file, and the attribute "Resources," which "specifies a resource" within the file "that appears to be compressed or encrypted."

Deep Learning Malware Detection

With the new deep learning model, we are able to perform a signatureless pre-execution evaluation of any executable file and determine if it is malware, potentially unwanted software, or a legitimate application.

At Sophos we've taken a unique approach to our security machine learning capabilities: we've invested heavily in deep neural network technology over more prevalent methods that, while still dominant in the security industry, are being rapidly abandoned by the machine learning computer science community.

How does Intercept X detect malicious executable files?

Instead of performing a signature and heuristic scan as traditional antivirus does, deep neural networks are able to extract millions of features from a file and determine if it is malicious before the program executes. The deep learning model learns what to look for in the code, how adversaries evade detection, how they build their software, and how the software plans to deploy and run. This information is evaluated by a multi-stage deep learning algorithm to determine how similar the software is to malware or potentially unwanted software. Depending on the score it is classified as malicious, potentially unwanted, or legitimate. It does all of this in about 20 milliseconds with a model that is under 20MB in size.

(*See* WBR_SOP000282, https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

Machine learning analysis

Under **Machine learning analysis**, you can see full results of our analysis.

Attributes shows a comparison of the file's attributes with those in millions of known bad and known good files. This enables you to determine how suspicious each attribute is and therefore whether the file is likely to be good or bad. You may see the following attributes:

- **Imports** describes the functionality that the file uses from external DLLs.
- **Strings** describes the most significant strings in the file.

- **Compilers** specifies what was used to compile the source code, for example C++, Delphi, Visual Basic, .NET.
- **Mitigation** describes techniques that the file uses to avoid being exploited.
- **Resources** specifies a resource that seems to be compressed or encrypted.
- **Summary** often relates to build or compilation dates, for example.
- **Packer** often specifies something of note about a specific section of the file, for example a suspicious section name or the fact that a section is both writable and executable.
- **Peid** refers to the output of PEiD, a third-party tool that scans a PE file against various malware signatures.
- **Btcaddress** shows any valid Bitcoin address that is found in the file.
- **Findcrypt** shows any suspicious cryptographic constants.

Code similarity shows a comparison of the file with millions of known bad and known good files, and lists the closest matches. Other matches count toward the result and may affect the rating for the file. The more bad files the file matches and the more closely it matches them, the more suspicious the file is.

File/path shows a comparison of the file's path with that of millions of known bad and known good files. If the file's path more closely matches the path of known bad files, the file is more likely to be suspicious. The path and file name used for comparison are either yours (if you requested the latest intelligence) or those from the last customer who sent us a file. We hide sensitive information in other customers' paths.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html.)

359. The Accused Products also comprise *at least one processor configured to perform operation of identifying an executable file to be evaluated* and *extracting a plurality of static data points from the executable file*. For example, Intercept X's "Deep Learning" technology, "extract[s] millions of features from a file and determine if it is malicious before the program executes." These extracted features include the most "significant strings found in the file," the attribute "Findcrypt," which "shows any suspicious cryptographic constants" found within the file, and the attribute "Resources," which "specifies a resource" within the file "that appears to be compressed or encrypted."

Deep Learning Malware Detection

With the new deep learning model, we are able to perform a signatureless pre-execution evaluation of any executable file and determine if it is malware, potentially unwanted software, or a legitimate application.

At Sophos we've taken a unique approach to our security machine learning capabilities: we've invested heavily in deep neural network technology over more prevalent methods that, while still dominant in the security industry, are being rapidly abandoned by the machine learning computer science community.

How does Intercept X detect malicious executable files?

Instead of performing a signature and heuristic scan as traditional antivirus does, deep neural networks are able to extract millions of features from a file and determine if it is malicious before the program executes. The deep learning model learns what to look for in the code, how adversaries evade detection, how they build their software, and how the software plans to deploy and run. This information is evaluated by a multi-stage deep learning algorithm to determine how similar the software is to malware or potentially unwanted software. Depending on the score it is classified as malicious, potentially unwanted, or legitimate. It does all of this in about 20 milliseconds with a model that is under 20MB in size.

(*See* WBR_SOP000282, https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

Machine learning analysis

Under **Machine learning analysis**, you can see full results of our analysis.

Attributes shows a comparison of the file's attributes with those in millions of known bad and known good files. This enables you to determine how suspicious each attribute is and therefore whether the file is likely to be good or bad. You may see the following attributes:

- **Imports** describes the functionality that the file uses from external DLLs.
- **Strings** describes the most significant strings in the file.
- **Compilers** specifies what was used to compile the source code, for example C++, Delphi, Visual Basic, .NET.
- **Mitigation** describes techniques that the file uses to avoid being exploited.
- **Resources** specifies a resource that seems to be compressed or encrypted.
- **Summary** often relates to build or compilation dates, for example.

- **Packer** often specifies something of note about a specific section of the file, for example a suspicious section name or the fact that a section is both writable and executable.
- **Peid** refers to the output of PEiD, a third-party tool that scans a PE file against various malware signatures.
- **Btcaddress** shows any valid Bitcoin address that is found in the file.
- **Findcrypt** shows any suspicious cryptographic constants.

Code similarity shows a comparison of the file with millions of known bad and known good files, and lists the closest matches. Other matches count toward the result and may affect the rating for the file. The more bad files the file matches and the more closely it matches them, the more suspicious the file is.

File/path shows a comparison of the file's path with that of millions of known bad and known good files. If the file's path more closely matches the path of known bad files, the file is more likely to be suspicious. The path and file name used for comparison are either yours (if you requested the latest intelligence) or those from the last customer who sent us a file. We hide sensitive information in other customers' paths.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/ProcessDetails.html; WBR_SOP000234, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ProcessDetails/index.html.)

360. The Accused Products also comprise a processor configured to perform operation of *generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files*. For example, Intercept X employs "Deep Learning models" (*i.e.*, advanced machine learning models) that have been trained on "hundreds of millions of samples" to classify the features extracted from a file to "determine if a file is benign or malicious…before the file executes." Such samples include "Potentially Unwanted Applications" such as adware. Before using a deep learning model to classify the features extracted from a file, Intercept X creates a "vector of information" that translates those features into data that the model can "intake" and "process."

Performance: Sophos' Deep Learning technology is incredibly fast. In less than 20 milliseconds the model is able to extract millions of features from a file, conduct a deep analysis, and determine if a file is benign or malicious. This entire process happens before the file executes.

SophosLabs: One of the most important aspects to any model is the data that used for training. Our team of data scientists are part of the SophosLabs group, granting them access to hundreds of millions of samples. This allows them to create the best possible predictions in our models. The integration between the two groups also leads to better data labeling (and therefore better modeling). The bi-directional sharing of threat intelligence and real-world feedback between the team of data scientists and threat researchers continuously improves the accuracy of our models.

(*See* WBR_SOP000300, https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophos-intercept-x-deep-learning-dsna.pdf.)

Feature Engineering in Machine Learning

Before creating a machine learning model, it's important to prepare our data. Preparing the data requires translating it into a language our model can understand. This is referred to as feature engineering.

Artificial neural network models intake data as a vector of information, so simply feeding the model a URL – which is not in the language of a vector – means that the model can't process it without some manipulation. There are countless ways that samples can be translated into features, though it takes some domain knowledge to do so. Using the URL example again, one way to translate a URL into a usable language is through a combination of ngramming and hashing. Ngrams are a popular method in DNA sequencing research. For example, the results of a three-gram ngram for the URL "https://sophos.com/company/careers.aspx" would be:

['htt', 'ttp', 'tps', 'ps:', 's:/', '://', '//s', '/so', 'sop', 'oph', 'pho', 'hos', 'os.', 's.c', '.co', 'com', 'om/', 'm/c', '/co', 'com', 'omp', 'mpa', 'pan', 'any', 'ny/', 'y/c', '/ca', 'car', 'are', 'ree', 'eer', 'ers', 'rs.', 's.a', '.as', 'asp', 'spx']

Once the ngrams are calculated, we need to translate them into a numerical representation. This can be done through a hashing mechanism. We will create an n-length long vector – say 1000 – and hash each ngram using a hashing algorithm. The resulting number from the hash of a particular ngram will be the index of which we will add 1. For example, if the first ngram 'htt' results in a hash of three and our vector is five units long, the result would be [0, 0, 1, 0, 0]. We continue this process for every ngram and for every URL until we have the list of URLs completely transformed into individual n-length vectors. When using this method for our toy model, these vectors will be 1,000 units long.

(*See* WBR_SOP000302, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/machine-learning-how-to-build-a-better-threat-detection-model.pdf?cmp=70130000001xKqzAAE.)

2.1 MLP Model

An MLP is a class of feedforward ANN with an input layer, an output layer, and one or more hidden layers between them. Each node in one layer fully connects to every node in the following layer. Except for nodes in the input layer, each node is a neuron (or processing element) with a nonlinear activation function associated with a scalar weight which is adjusted during training. An MLP is a supervised learning algorithm that learns to identify data patterns for classification or regression. In order to carry out the learning process, one needs to extract a digital representation $\boldsymbol{x}$ of a given object or event that needs to be fed into the MLP. The learning task becomes to find a multidimensional function $\Phi(\cdot)$ between input $\boldsymbol{x}$ and target $\boldsymbol{y}$

$$\boldsymbol{y} \cong \Phi(\boldsymbol{x}) \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^N$, a real-valued input feature vector $\boldsymbol{x} = [x_1, \dots, x_N]^T$ in an N dimensional feature space, with $(\cdot)^T$ denoting the transpose operation. Similarly, $\boldsymbol{y} \in \mathbb{R}^M$, a real-valued target classification vector $\boldsymbol{y} = [y_1, \dots, y_M]^T$ in an M dimensional classification space. In other words, an MLP learning process is to find $\Phi(\cdot)$

(*See* WBR_SOP000318, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophos-black-hat-2018-technicalpaper.pdf.)

Generic ML PUA detections

Potentially Unwanted Application (PUA) is a term used to describe applications that, while not malicious, are generally considered unsuitable for business networks.

A Generic ML PUA detection is generated by Sophos Intercept X's Machine Learning (ML) engine, also referred to by the specific Sophos approach Deep Learning and is designed to detect PUAs in PE (Portable Executable) files such as:

- .exe
- .sys
- .dll
- .pif
- .scr
- and many more

If a detection of this type has been received, it is because Sophos has detected a file on the computer that our Deep Learning threat model has decided is a PUA. This is a pre-execution detection meaning the file was detected before it was able to be run.

Major PUA classifications

The major PUA classifications are:

- adware
- dialer
- non-malicious spyware
- remote administration tools
- hacking tools

(*See* WBR_SOP000318, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophos-black-hat-2018-technicalpaper.pdf.)

361. The feature vector is "evaluated by a multi-stage deep learning algorithm to determine how similar the software is to malware or potentially unwanted software. Depending on the score [the file] is classified as malicious, potentially unwanted, or legitimate."

Deep Learning Malware Detection

With the new deep learning model, we are able to perform a signatureless pre-execution evaluation of any executable file and determine if it is malware, potentially unwanted software, or a legitimate application.

At Sophos we've taken a unique approach to our security machine learning capabilities: we've invested heavily in deep neural network technology over more prevalent methods that, while still dominant in the security industry, are being rapidly abandoned by the machine learning computer science community.

How does Intercept X detect malicious executable files?

Instead of performing a signature and heuristic scan as traditional antivirus does, deep neural networks are able to extract millions of features from a file and determine if it is malicious before the program executes. The deep learning model learns what to look for in the code, how adversaries evade detection, how they build their software, and how the software plans to deploy and run. This information is evaluated by a multi-stage deep learning algorithm to determine how similar the software is to malware or potentially unwanted software. Depending on the score it is classified as malicious, potentially unwanted, or legitimate. It does all of this in about 20 milliseconds with a model that is under 20MB in size.

(*See* WBR_SOP000282, https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

362. The Accused Products generate a feature vector wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range. For example, Intercept X employs "feature selection" to only "keep relevant features before feeding them into" its deep learning models, "identifying and removing as much noisy and redundant information as possible from extracted features."

A crucial step in an ML workflow is feature extraction, which can be hand-crafted-based on human expertise, or automatically learned by training modern deep learning models such as convolutional neural networks (CNNs). It is natural to believe that more extracted features can provide better characterization of a learning task and more discriminating power. However, increasing the dimension of the feature vector could result in feature redundancy and noise. Redundant and irrelevant features can cause performance deterioration of an ML model with overfitting and generalization problems. Additionally, excessively increased number of features could significantly slow down a learning process. Therefore, it is of fundamental importance to only keep relevant features before feeding them into an ML model, which leads to requiring feature selection (or feature dimensionality reduction). Feature selection can be seen as the process of identifying and removing as much noisy and redundant information as possible from extracted features.

(*See* WBR_SOP000329, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophoslabs-machine-learning-tp.pdf.)

363. The Accused Products constitute at least one processor configured to perform the operation of *evaluating the feature vector using a machine learning model to determine whether the executable file is harmful*. For example, and as explained above, Intercept X employs "Deep Learning models" (*i.e.*, advanced machine learning models) that have been trained on "hundreds of millions of samples" to classify the features extracted from a file to "determine if a file is benign or malicious…before the file executes." Before using a deep learning model to classify the features extracted from a file, Intercept X creates a "vector of information" that translates those features into data that the model can "intake" and "process."

> **Performance:** Sophos' Deep Learning technology is incredibly fast. In less than 20 milliseconds the model is able to extract millions of features from a file, conduct a deep analysis, and determine if a file is benign or malicious. This entire process happens before the file executes.
>
> **SophosLabs:** One of the most important aspects to any model is the data that used for training. Our team of data scientists are part of the SophosLabs group, granting them access to hundreds of millions of samples. This allows them to create the best possible predictions in our models. The integration between the two groups also leads to better data labeling (and therefore better modeling). The bi-directional sharing of threat intelligence and real-world feedback between the team of data scientists and threat researchers continuously improves the accuracy of our models.

(*See* WBR_SOP000300, https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophos-intercept-x-deep-learning-dsna.pdf.)

Feature Engineering in Machine Learning

Before creating a machine learning model, it's important to prepare our data. Preparing the data requires translating it into a language our model can understand. This is referred to as feature engineering.

Artificial neural network models intake data as a vector of information, so simply feeding the model a URL – which is not in the language of a vector – means that the model can't process it without some manipulation. There are countless ways that samples can be translated into features, though it takes some domain knowledge to do so. Using the URL example again, one way to translate a URL into a usable language is through a combination of ngramming and hashing. Ngrams are a popular method in DNA sequencing research. For example, the results of a three-gram ngram for the URL "https://sophos.com/company/careers.aspx" would be:

['htt', 'ttp', 'tps', 'ps:', 's:/', '://', '//s', '/so', 'sop', 'oph', 'pho', 'hos', 'os.', 's.c', '.co', 'com', 'om/', 'm/c', '/co', 'com', 'omp', 'mpa', 'pan', 'any', 'ny/', 'y/c', '/ca', 'car', 'are', 'ree', 'eer', 'ers', 'rs.', 's.a', '.as', 'asp', 'spx']

Once the ngrams are calculated, we need to translate them into a numerical representation. This can be done through a hashing mechanism. We will create an n-length long vector – say 1000 – and hash each ngram using a hashing algorithm. The resulting number from the hash of a particular ngram will be the index of which we will add 1. For example, if the first ngram 'htt' results in a hash of three and our vector is five units long, the result would be [0, 0, 1, 0, 0]. We continue this process for every ngram and for every URL until we have the list of URLs completely transformed into individual n-length vectors. When using this method for our toy model, these vectors will be 1,000 units long.

(*See* WBR_SOP000302, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/machine-learning-how-to-build-a-better-threat-detection-model.pdf?cmp=70130000001xKqzAAE.)

364. The feature vector is "evaluated by a multi-stage deep learning algorithm to determine how similar the software is to malware or potentially unwanted software. Depending on the score [the file] is classified as malicious, potentially unwanted, or legitimate."

Deep Learning Malware Detection

With the new deep learning model, we are able to perform a signatureless pre-execution evaluation of any executable file and determine if it is malware, potentially unwanted software, or a legitimate application.

At Sophos we've taken a unique approach to our security machine learning capabilities: we've invested heavily in deep neural network technology over more prevalent methods that, while still dominant in the security industry, are being rapidly abandoned by the machine learning computer science community.

How does Intercept X detect malicious executable files?

Instead of performing a signature and heuristic scan as traditional antivirus does, deep neural networks are able to extract millions of features from a file and determine if it is malicious before the program executes. The deep learning model learns what to look for in the code, how adversaries evade detection, how they build their software, and how the software plans to deploy and run. This information is evaluated by a multi-stage deep learning algorithm to determine how similar the software is to malware or potentially unwanted software. Depending on the score it is classified as malicious, potentially unwanted, or legitimate. It does all of this in about 20 milliseconds with a model that is under 20MB in size.

(*See* WBR_SOP000329, https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophoslabs-machine-learning-tp.pdf.)

365. Each claim in the '869 Patent recites an independent invention. Neither claim 46, described above, nor any other individual claim is representative of all claims in the '869 Patent.

366. Defendant has been aware of the '869 Patent since at least the filing of this Second Amended Complaint. Further, Plaintiffs have marked its products with the '869 Patent, including on its web site, since at least July 2020.

367. Defendant directly infringes at least claim 46 of the '869 Patent, either literally or under the doctrine of equivalents, by performing the steps described above. For example, on information and belief, Defendant makes and uses the claimed system in an infringing manner as

described above by running the Sophos security software and corresponding systems to protect its own computer and network operations. On information and belief, Defendant also performs the claimed method as described above when testing the operation of the Accused Products and corresponding systems. As another example, Defendant performs the claimed method as described above when providing or administering services to third parties, customers, and partners using the Accused Products.

368. Defendant's partners, customers, and end users of the Accused Products and corresponding systems and services directly infringe at least claim 46 of the '869 Patent, literally or under the doctrine of equivalents, at least by making and using the Accused Products and corresponding systems and services, as described above.

369. Defendant has actively induced and is actively inducing infringement of at least claim 46 of the '869 Patent with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b). For example, Sophos encourages and induces customers with specific intent to use Sophos's security software in a manner that infringes claim 46 of the '869 Patent at least by offering and providing software that software that and/or hardware that embodies the system that infringes claim 46 when installed and operated by the customer, and by engaging in activities relating to selling, marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including the activities described below.

370. Defendant encourages, instructs, directs, and/or requires third parties—including its certified partners and/or customers to perform the claimed method using the software, services, and systems in infringing ways, as described above.

371. Defendant further encourages and induces its customers to infringe claim 46 of the

'869 Patent: 1) by making its security services available on its website, providing applications that allow users to access those services, widely advertising those services, and providing technical support and instructions to users, and 2) through activities relating to marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including its Sophos security software, and services in the United States. (*See* https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx; WBR_SOP000525, https://partners.sophos.com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1; WBR_SOP000209, https://secure2.sophos.com/en-us/security-news-trends/whitepapers/gated-wp/cybersecurity-system-buyers-guide.aspx%23formFrame; *see also* WBR_SOP000214, https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophos-synchronized-security-ds.pdf.)

372. For example, on information and belief, Defendant shares instructions, guides, and manuals, which advertise and instruct third parties on how to use the software as described above, including at least customers and partners. (*Id.*) On further information and belief, Defendant provides customer service and technical support to purchasers of the Accused Products and corresponding system and services, which directs and encourages customers to perform certain actions that use the Accused Products in an infringing manner. (*Id.*)

373. Defendant and/or its partners recommend and sell the Accused Products and provide technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Defendant and/or one of its partners, which obligates each customer to perform certain actions in order to use the Accused Products. Further, in order to receive the benefit of Defendant's or its partner's continued technical support and their specialized

knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that infringes the ‘869 Patent.

374. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Product remains operational for each customer through ongoing technical support, on information and belief, Defendant and/or Sophos’s partners affirmatively aid and abet each customer’s use of the Accused Products in a manner that infringes the ‘869 Patent.

375. Defendant also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the claimed methods, and not a staple article or commodity of commerce suitable for substantial non-infringing uses. Indeed, as shown above, the Accused Products and the example functionality described below have no substantial non-infringing uses but are specifically designed to practice the ‘869 Patent.

376. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the machine learning static analysis and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. For example, without extracting a set of static data points from an object, the Accused Products could not create feature vector that they could feed to their machine learning model. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same

reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ‘869 Patent, that functionality could not be performed.

377. Additionally, the accused functionality, including the machine learning static analysis and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without extracting and analyzing a set of static features from an object and calibrating a feature vector on the basis of that analysis, the Accused Products could not deploy their machine learning model to determine whether or not that object is malicious or benign. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ‘869 Patent, that functionality could not be performed.

378. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Defendant constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionality (including the machine learning static analysis) constitute a material part of the inventions claimed at least because such an attack chain is integral to the processes identified above (such as “extracting a plurality of static data points from an executable file…”; “generating a feature vector from the plurality of static data points…”; “wherein one or more features of the feature vector are selectively turned on or off…”; “evaluating the feature vector…”) as recited in the claims of the ‘869 Patent. None of these

products are staple goods—they are sophisticated and customized cyber security and malware detection products, methods, and systems.

379. Defendant's infringing actions have continued after the Second Amended Complaint was filed. Defendant had knowledge of the '869 Patent and of the specific acts that constitutes infringement of the '869 Patent at least based on the filing of the Second Amended Complaint, but has continued to engage in the infringing activity, including by selling, making, configuring, and installing the Accused Products and performing the accused functionality, and by engaging in activities that constituted infringement and contributory infringement as described above.

380. On information and belief, the infringing actions of each partner, customer, and/or end-user of the Accused Products are attributable to Defendant. For example, on information and belief, Defendant directs and controls the activities or actions of its partners or others in connection with the Accused Products by contractual agreement or otherwise requiring partners or others to provide information and instructions to customers who acquire the Accused Products which, when followed, results in infringement. Defendant further directs and controls the operation of devices executing the Accused Products by programming the software which, when executed by a customer or end user, infringes at least claim 46 of the'869 Patent.

381. Plaintiffs have suffered and continue to suffer damages, including lost profits, as a result of Defendant's infringement of the '869 Patent. Defendant is therefore liable to Plaintiffs under 35 U.S.C. § 284 for damages in an amount that adequately compensates Plaintiffs for Defendant's infringement, but no less than a reasonable royalty.

382. Plaintiffs will continue to suffer irreparable harm unless this Court preliminarily and permanently enjoins Defendant, its agents, employees, representatives, and all others acting

in concert with Defendant from infringing the ’869 Patent.

383. Defendant’s infringement of the ’869 Patent is knowing and willful. Defendant had actual knowledge of the ’ 869 Patent at least by the time Plaintiffs filed this Second Amended Complaint.

384. On information and belief, despite Defendant’s knowledge of the Asserted Patents, Defendant made the deliberate decision to sell products and services that they knew infringe the ’869 Patent. Defendant’s continued infringement of the ’869 Patent with knowledge of the ’ 869 Patent constitutes willful infringement.

385. Plaintiffs’ allegations of infringement, indirect infringement, and willful infringement with respect to this patent are further set forth in Plaintiffs’ Disclosure of Asserted Claims and Preliminary Infringement Contentions Pursuant to the Court’s Standing Order Governing Patent Proceedings served July 12, 2022.

NINTH CAUSE OF ACTION
(INFRINGEMENT OF THE ’244 PATENT)

386. Plaintiffs reallege and incorporate by reference the allegations of the preceding paragraphs of this Complaint.

387. Defendant has infringed and continues to infringe one or more claims of the ’244 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including features such as Intercept X, at least when used for their ordinary and customary purposes, practice each element of at least claim 1 of the ’244 Patent, as described below.

388. For example, claim 1 of the ’244 Patent recites:

A method for identifying an origin of suspected pestware activity on a computer, the method comprising:

monitoring, with a kernel-mode driver, activity on the computer;

generating an activity log on a file storage device of the computer from the kernel-mode driver;

receiving, from a user via an interface of the computer, a time of interest relating to a suspicion of pestware on the computer, wherein the time of interest includes a time interval;

issuing a timestamp after receiving the time of interest;

identifying, based upon the time of interest, indicia of pestware, wherein the identifying is initiated by the issuing the timestamp; and

accessing, using a hardware processor of the computer, at least a portion of a recorded history of externally networked sources that the computer received files from so as to identify, based at least in part upon the identified indicia of pestware, a reference to an identity of an externally networked source that is suspected of originating pestware;

wherein the recorded history of externally networked sources is stored on the file storage device.

389. The Accused Products perform each element of the method of claim 1 of the ’244 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform a method for *identifying an origin of suspected pestware activity on a computer,* as further explained below. For example, Intercept X allows users to “investigate detected threats (threat graphs) and search for new threats or security weaknesses.” It also monitors devices and fixes issues remotely. The threat graphs allow users to “investigate and clean up malware attacks” and “find out where an attack started, how it spread, and which processes or files it has affected.” The threat graph analysis provides a summary of the pestware activity on a computer including details of root cause and name of the device and its user. The threat graph analysis also shows chain of events in the malware infection.

Intercept X with XDR

Sophos Extended Detection and Response (XDR) lets you investigate detected threats (threat graphs) and search for new threats or security weaknesses. It also lets you monitor devices and fix issues remotely.

You can find most XDR features in **Threat Analysis Center**.

These features are available if you have an Intercept X license that includes Sophos XDR.

Threat graphs

Threat graphs let you investigate and clean up malware attacks.

You can find out where an attack started, how it spread, and which processes or files it has affected.

For help, see Threat Graphs.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/XDR/index.html.)

Summary

The **Summary** shows you a summary of the threat, including these details:

- **Root cause**: Where the infection entered your system.
- **Possible data involved**: Files that might contain important data. Check them to see if data has been encrypted or stolen.
- **Where**: Name of the device and its user.
- **When**: Detection time and date.

Analyze

The **Analyze** tab shows the chain of events in the malware infection.

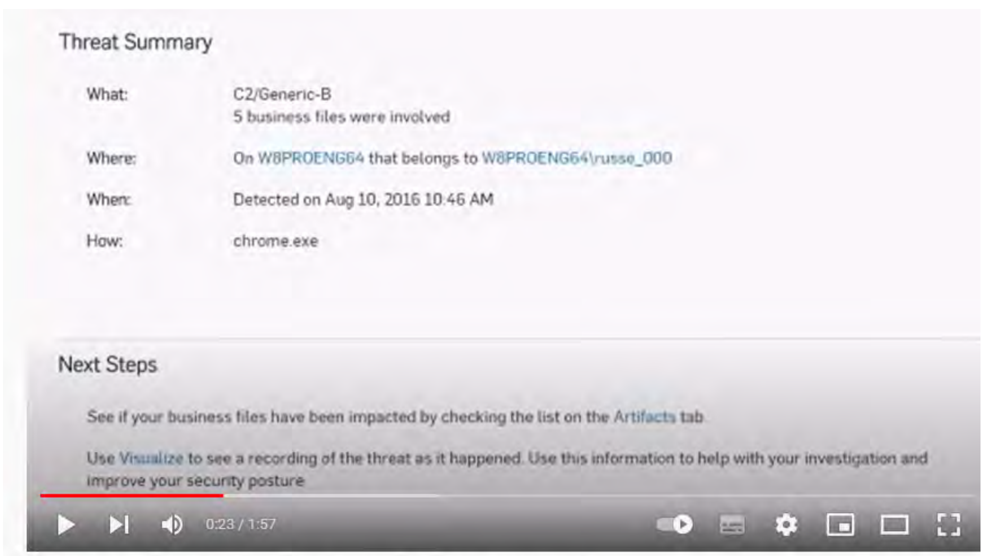
A menu on the right of the tab lets you choose how much detail you see:

- **Show direct path**: This shows the chain of directly-involved items between the root cause and the item where the infection was detected (the "beacon").
- **Show full graph**: This shows the root cause, beacon, affected artifacts (applications, files, keys), the path of the infection (shown by arrows), and how the infection occurred. This is the default setting.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

Among other features, Intercept X gives root-cause analysis to quickly get to the bottom of an attack and know what to do next.

(*See* https://news.sophos.com/en-us/2016/10/28/watch-now-sophos-intercept-x-root-cause-analysis-in-two-minutes/.)



Root Cause Analysis RCA in 2 Minutes | Sophos Intercept X

(*See* https://www.youtube.com/watch?v=AOsjUjp4P7Q.)

390. The Accused Products perform a method that includes *monitoring, with a kernel-*

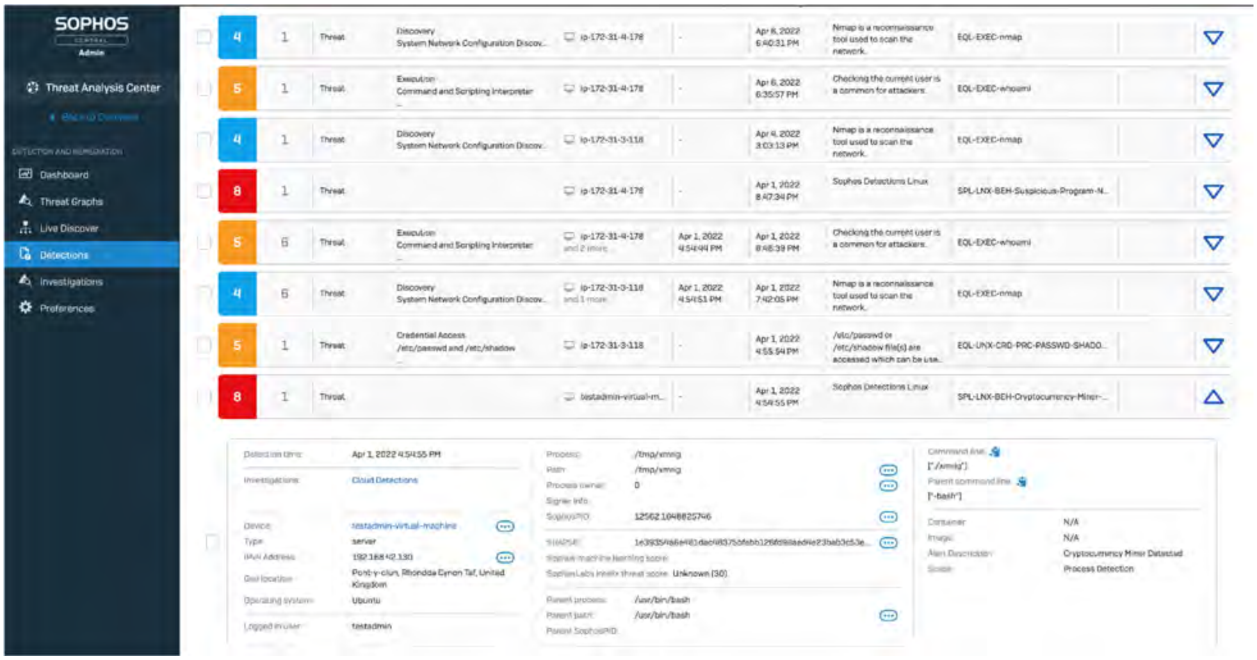
mode driver, activity on the computer and generating an activity log on a file storage device of the computer from the kernel-mode driver. For example, Intercept X includes enhanced visibility for Linux hosts and container workloads. Intercept X's "Detections" dashboard includes some new detections that leverages analytics around attacker behavior, from initial access (including application and system exploitation) to privilege escalation, defense evasion, data collection, exfiltration, and many others. Specifically, one of the new detections is "kernel exploits" detection, which "highlights if internal kernel functions are being tampered" with on a host.

Included as part of the Sophos XDR Detections dashboard, the new detections leverage analytics around attacker behavior, from initial access (including application and system exploitation) to privilege escalation, defense evasion, data collection, exfiltration, and many others.

Example new detections include:

- Container escapes: Identifies attackers escalating privileges from container access to move across to the container host
- Cryptominers: Detects program names or arguments commonly associated with cryptocurrency miners
- Data destruction: Alerts that an attacker may be trying to delete indicators of compromise that are part of an ongoing investigation
- Kernel exploits: Highlights if internal kernel functions are being tampered with on a host

(*See* https://news.sophos.com/en-us/2022/04/14/enhanced-linux-and-container-security-for-sophos-intercept-x-for-server/)



These host and container threat detections are automatically converted into an investigation, with an AI-prioritized risk score for each detection. Scores are then color-coded by risk level, enabling security teams to quickly identify where they should focus to increase efficiency. Integrated Live Response further establishes a secure command line terminal to hosts for rapid remediation.

(*See* https://news.sophos.com/en-us/2022/04/14/enhanced-linux-and-container-security-for-sophos-intercept-x-for-server/.)

391. Intercept X also generates a "threat graph" from the detections made by its kernel-mode driver. For example, the "Graph record" tab shows the "history of the threat graph, from its creation by Sophos or the admin." Moreover, the "artifacts list" of the threat graph shows all "the affected items of a malware attack, including business files, processes, registry keys, or IP addresses."

Graph record

The **Graph record** tab shows the history of the threat graph, from its creation by Sophos or the admin. You can post comments to record actions that have been taken and other relevant information.

Process details

When you click an affected item, you see the **Process details** pane. If someone has already submitted the file to Sophos, you see the latest threat intelligence.

If the file hasn't been submitted, or you want to see if there's any updated intelligence, click **Request latest intelligence**.

This shows the latest information about the file's global reputation and whether you need to investigate.

Artifacts list

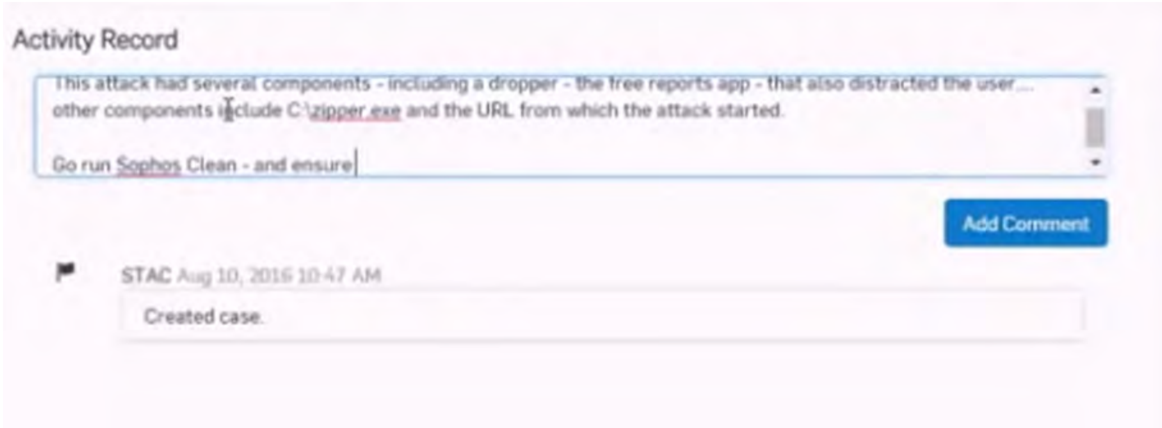
This is a list below the diagram of the malware attack. It shows all the affected items, for example business files, processes, registry keys, or IP addresses.

You can export a comma separated (CSV) file containing a list of the affected artifacts, by clicking on **Export to CSV** at the top right of the tab.

The list shows:

- **Name**: Click the name to see more information in a details pane.
- **Type**: The type of artifact, such as a business file or a registry key.
- **Reputation**
- **Time logged**: The time and date a process was accessed.
- **Interactions**

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html#analyze.)



(*See* https://www.youtube.com/watch?v=AOsjUjp4P7Q.)

392. Furthermore, Intercept X generates a "forensic snapshot" of data from a Sophos log of a given device's activity and collected by the kernel-mode driver. The "forensic snapshot" includes activities for a certain time period.

Create forensic snapshot

You can create a "forensic snapshot" of data from the device. This gets data from a Sophos log of the device's activity and saves it on that device. For more information on forensic snapshots see Forensic snapshots.

You can also save it in the Amazon Web Services (AWS) S3 bucket you specify. You can then do your analysis.

You'll need a converter (which we provide) to read the data.

> **Note**
>
> You can choose how much data you want in snapshots and where to upload them. To do this, go to **Global Settings** > **Forensic Snapshots**. These options may not be available for all customers yet.

To create a snapshot, do as follows:

1. Go to a threat graph's **Analyze** tab.

   Alternatively, on the details page of the device, open the **Status** tab.
2. Click **Create forensic snapshot**.
3. Follow the steps in Upload a forensic snapshot to an AWS S3 bucket.

You can find the snapshots you generated in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Forensic Snapshots\`.

Snapshots generated from detections are in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Saved Data\`.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

Forensic snapshots

Forensic snapshots get data from a Sophos log of a computer's activity so that you can do your own analysis.

You can create a forensic snapshot from a threat graph or from the **Status** tab in a device's details page.

Go to **Global Settings** > **Forensic Snapshots**.

You can configure how much data you want in your snapshots and where you want to put them.

Note

The configuration options may not be available for all customers yet.

Set the time period for the forensic snapshot

By default, a snapshot includes data for the previous two weeks.

Here you can set a different time period or choose to include all the available data.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/GlobalSettings/ForensicSnapshots/index.html.)

393. The Accused Products perform a method that includes *receiving, from a user via an interface of the computer, a time of interest relating to a suspicion of pestware on the computer, wherein the time of interest includes a time interval*. As illustrated in the screenshots below, Intercept X uses Live discover queries to search devices for signs of threats within a selected time period. Exemplary time periods include "last 24 hours," "last 7 days," and "last 30 days."

Live Discover

Live Discover allows you to check the devices that Sophos Central is managing, look for signs of a threat, or assess compliance.

You can use Live Discover queries to search devices for signs of threats that haven't been detected by other Sophos features. For example:

- Unusual changes to the registry.
- Failed authentications.
- A process running that is very rarely run.

You can also search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere, or if a user reports suspicious behavior on their device.

You can also check the compliance of each device. For example, you can search for out-of-date software or browsers with insecure settings.

In this video, we will demonstrate how to hunt for threat activity in your environment with CrowdStrike Falcon. First, we see how you can use Falcon to search for indicators of compromise (IOCs). Then we take a broader look at how we can use built-in dashboards to quickly uncover and investigate suspicious activity. Finally, we see how power users can craft precise queries to search for new and unique attacker tactics, techniques and procedures (TTPs) on data stored in the CrowdStrike Threat Graph.

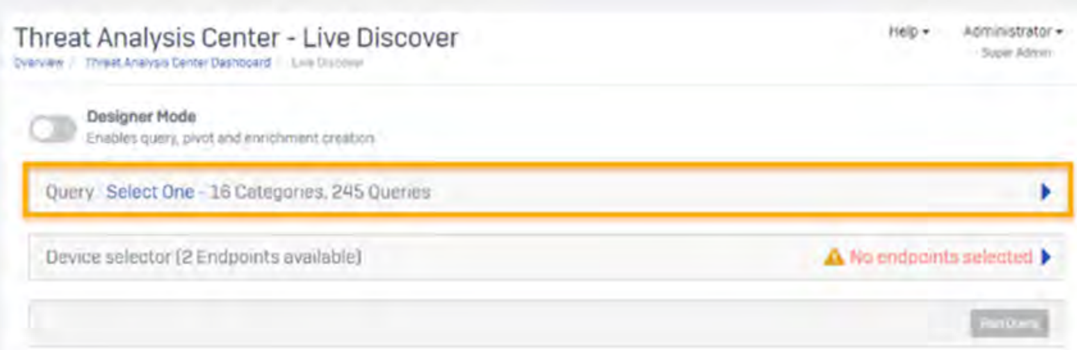
(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/LiveDiscover/index.html.)

Select query

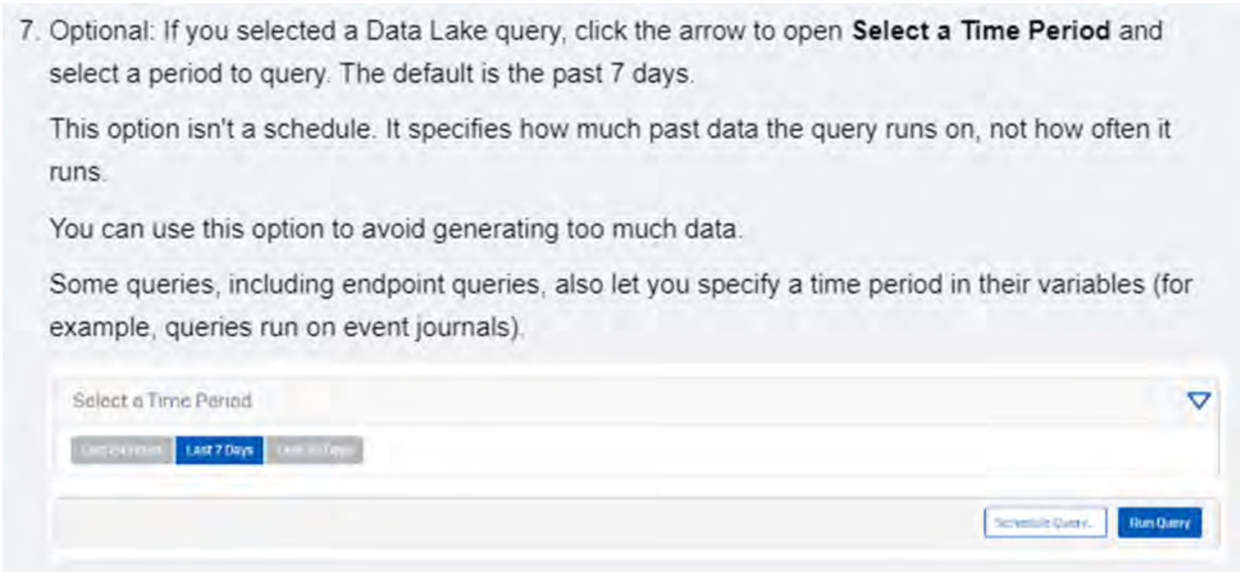
To select a pre-prepared query, do as follows:

1. Go to **Threat Analysis Center** and click **Live Discover**.
2. In **Live Discover**, open the **Query** section (if it isn't already open).

   Designer Mode lets you edit or create queries. You don't need to turn it on if you're using our pre-prepared queries.



(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/LiveDiscover/index.html.)



7. Optional: If you selected a Data Lake query, click the arrow to open **Select a Time Period** and select a period to query. The default is the past 7 days.

   This option isn't a schedule. It specifies how much past data the query runs on, not how often it runs.

   You can use this option to avoid generating too much data.

   Some queries, including endpoint queries, also let you specify a time period in their variables (for example, queries run on event journals).

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/LiveDiscover/index.html.)

394. Furthermore, Intercept X generates a "forensic snapshot" of data from a Sophos log of a given device's activity and collected by the kernel-mode driver. The "forensic snapshot" includes activities for a certain time period.

Create forensic snapshot

You can create a "forensic snapshot" of data from the device. This gets data from a Sophos log of the device's activity and saves it on that device. For more information on forensic snapshots see Forensic snapshots.

You can also save it in the Amazon Web Services (AWS) S3 bucket you specify. You can then do your analysis.

You'll need a converter (which we provide) to read the data.

Note

You can choose how much data you want in snapshots and where to upload them. To do this, go to **Global Settings** > **Forensic Snapshots**. These options may not be available for all customers yet.

To create a snapshot, do as follows:

1. Go to a threat graph's **Analyze** tab.

   Alternatively, on the details page of the device, open the **Status** tab.

2. Click **Create forensic snapshot**.
3. Follow the steps in Upload a forensic snapshot to an AWS S3 bucket.

You can find the snapshots you generated in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Forensic Snapshots\`.

Snapshots generated from detections are in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Saved Data\`.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

Forensic snapshots

Forensic snapshots get data from a Sophos log of a computer's activity so that you can do your own analysis.

You can create a forensic snapshot from a threat graph or from the **Status** tab in a device's details page.

Go to **Global Settings** > **Forensic Snapshots**.

You can configure how much data you want in your snapshots and where you want to put them.

> **Note**
>
> The configuration options may not be available for all customers yet.

Set the time period for the forensic snapshot

By default, a snapshot includes data for the previous two weeks.

Here you can set a different time period or choose to include all the available data.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/GlobalSettings/ForensicSnapshots/index.html.)

395. The Accused Products perform a method that includes "*issuing a timestamp after receiving the time of interest*." For example, Intercept X's Threat Analysis Center includes a detection feature to allow users to investigate threats over their time of interest, such as last 24 hours," "last 7 days," and "last 30 days." Detections identify unusual or suspicious activity on devices that have not yet been designated as malicious. The detection list includes information about when the detection was "First Seen" and "Last Seen", which timestamp the detected activities.

Detections

Detections show you activity that you might need to investigate.

To see detections, go to **Threat Analysis Center** > **Detections**.

Detections identify activity on your devices that's unusual or suspicious but hasn't been blocked. They're different from events where we detect and block activity that we already know to be malicious.

We generate detections based on data that devices upload to the Sophos Data Lake.

We check that data against threat classification rules. When there's a match, we show a detection.

This page tells you how to use detections to look for potential threats.

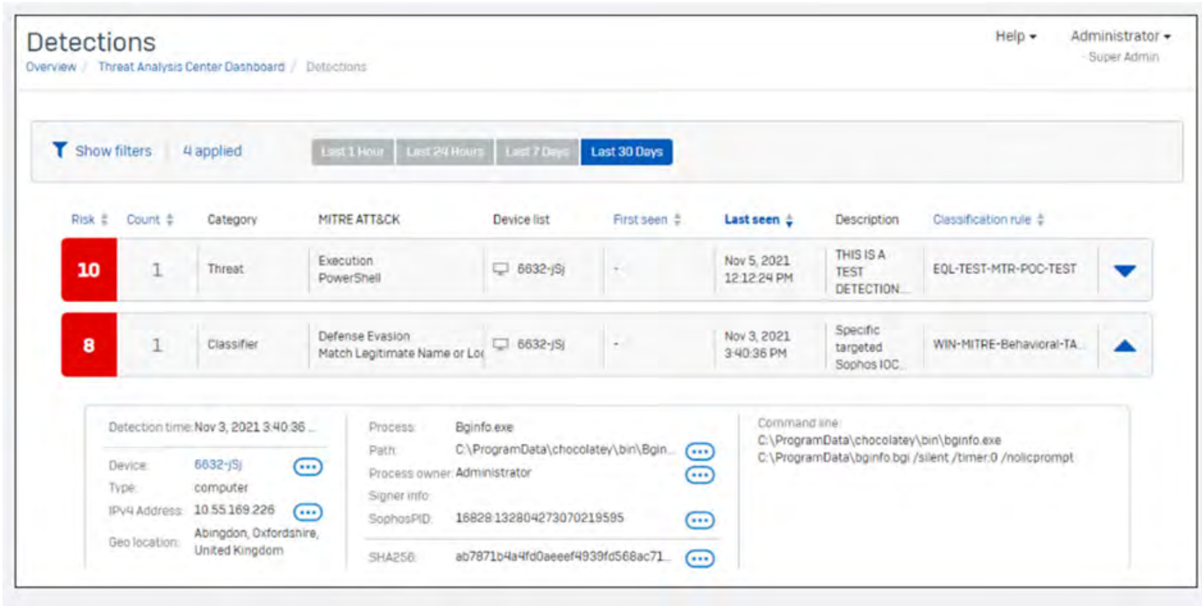
> **Note**
>
> **Investigations** can automatically group related detections together for more advanced analysis. See Investigations.

View detection details

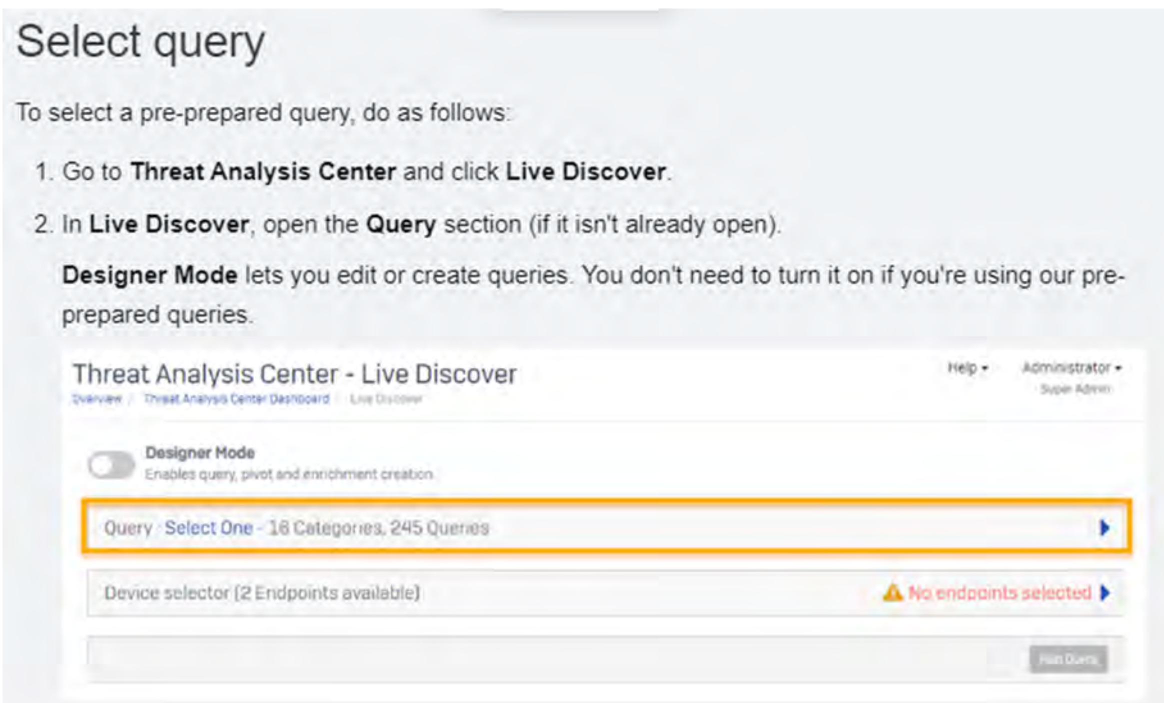
To see detections, go to **Threat Analysis Center** > **Detections**.

We group detections according to the rule they matched and the date. The detection list shows the following:

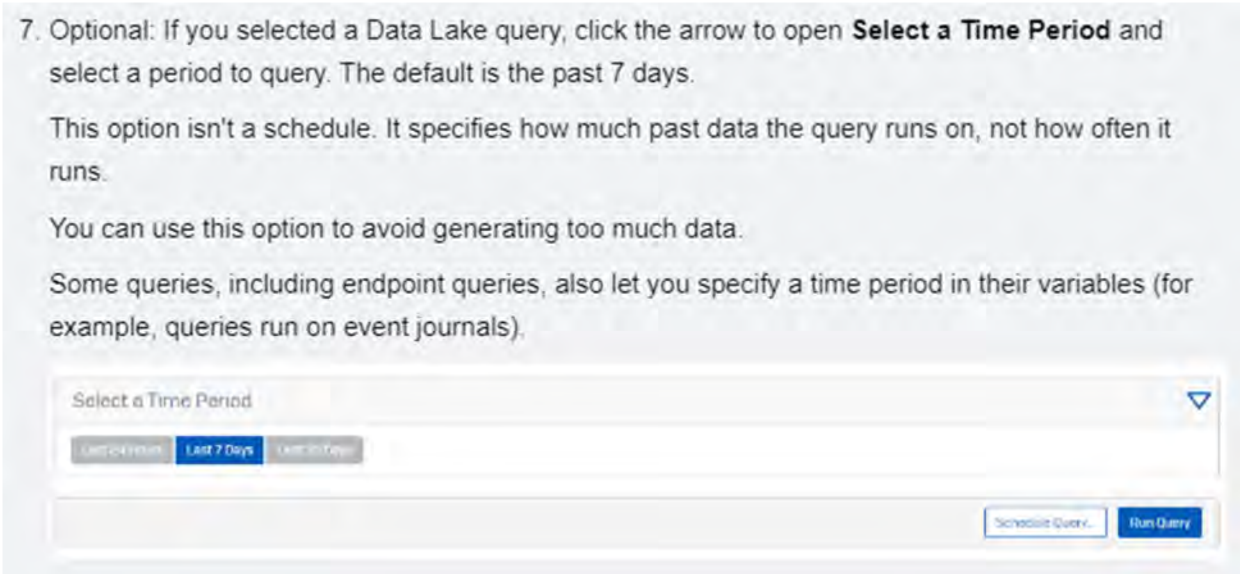
- **Risk**. Risk is on a scale of 1 (lowest) to 10 (highest). With the default settings, we only show detections with a score of 7 or more. Use the score to prioritize investigations.
- **Classification rule**. The name of the rule that was matched.
- **Count**. Number of times the classification rule has been matched on a certain day.
- **Device list**. The device where the rule was last matched and the number of other devices with the same detection that day.
- **First seen** and **Last seen**. The first and last detections based on the classification rule that day.
- **Description**. What the rule identifies.
- **Mitre ATT&CK**. The corresponding Mitre ATT&CK Tactic and Technique.



(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/Detections/index.html.)



(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/LiveDiscover/index.html.)


7. Optional: If you selected a Data Lake query, click the arrow to open **Select a Time Period** and select a period to query. The default is the past 7 days.

This option isn't a schedule. It specifies how much past data the query runs on, not how often it runs.

You can use this option to avoid generating too much data.

Some queries, including endpoint queries, also let you specify a time period in their variables (for example, queries run on event journals).

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/LiveDiscover/index.html.)

396. The Accused Products perform a method that *includes identifying, based upon the time of interest, indicia of pestware, wherein the identifying is initiated by the issuing the timestamp*.

397. For example, Intercept X's Threat Analysis Center includes a detection feature to allow users to investigate threats detected over the "Last 7 days," "Last 24 Hours," and "Last Hour."

Detections

Detections show you activity that you might need to investigate.

To see detections, go to **Threat Analysis Center** > **Detections**.

Detections identify activity on your devices that's unusual or suspicious but hasn't been blocked. They're different from events where we detect and block activity that we already know to be malicious.

We generate detections based on data that devices upload to the Sophos Data Lake.

We check that data against threat classification rules. When there's a match, we show a detection.

This page tells you how to use detections to look for potential threats.

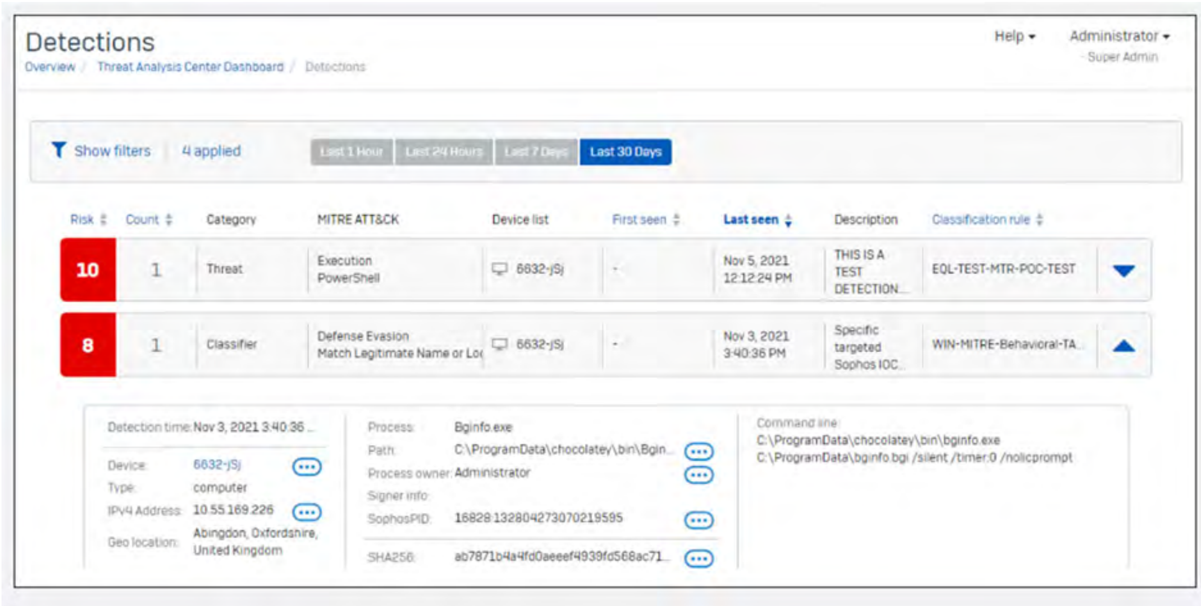
> **Note**
>
> **Investigations** can automatically group related detections together for more advanced analysis. See Investigations.

View detection details

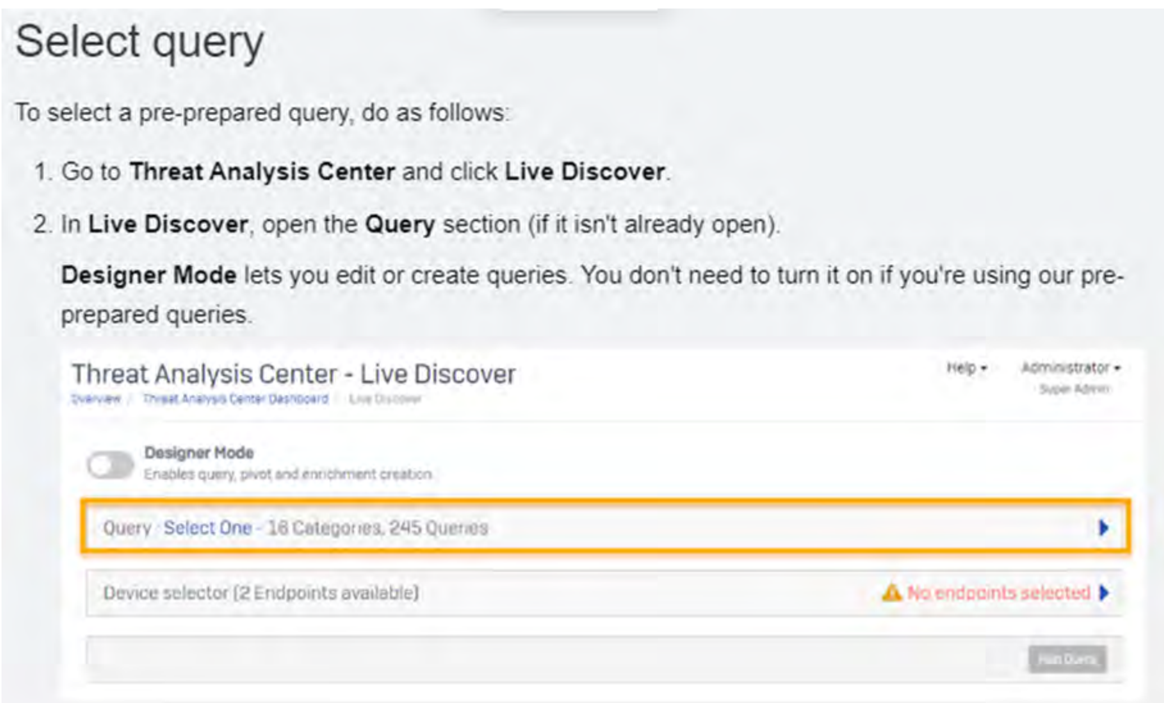
To see detections, go to **Threat Analysis Center** > **Detections**.

We group detections according to the rule they matched and the date. The detection list shows the following:

- **Risk**. Risk is on a scale of 1 (lowest) to 10 (highest). With the default settings, we only show detections with a score of 7 or more. Use the score to prioritize investigations.
- **Classification rule**. The name of the rule that was matched.
- **Count**. Number of times the classification rule has been matched on a certain day.
- **Device list**. The device where the rule was last matched and the number of other devices with the same detection that day.
- **First seen** and **Last seen**. The first and last detections based on the classification rule that day.
- **Description**. What the rule identifies.
- **Mitre ATT&CK**. The corresponding Mitre ATT&CK Tactic and Technique.



(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/Detections/index.html.)



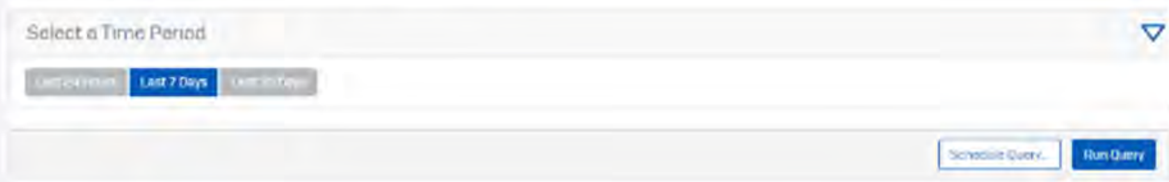
(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/LiveDiscover/index.html.)

7. Optional: If you selected a Data Lake query, click the arrow to open **Select a Time Period** and select a period to query. The default is the past 7 days.

   This option isn't a schedule. It specifies how much past data the query runs on, not how often it runs.

   You can use this option to avoid generating too much data.

   Some queries, including endpoint queries, also let you specify a time period in their variables (for example, queries run on event journals).



(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/LiveDiscover/index.html.)

398. Furthermore, Intercept X generates a "forensic snapshot" of data from a Sophos log of a given device's activity and collected by the kernel-mode driver. The "forensic snapshot" includes activities for a certain time period.

Create forensic snapshot

You can create a "forensic snapshot" of data from the device. This gets data from a Sophos log of the device's activity and saves it on that device. For more information on forensic snapshots see Forensic snapshots.

You can also save it in the Amazon Web Services (AWS) S3 bucket you specify. You can then do your analysis.

You'll need a converter (which we provide) to read the data.

> **Note**
>
> You can choose how much data you want in snapshots and where to upload them. To do this, go to **Global Settings** > **Forensic Snapshots**. These options may not be available for all customers yet.

To create a snapshot, do as follows:

1. Go to a threat graph's **Analyze** tab.

   Alternatively, on the details page of the device, open the **Status** tab.
2. Click **Create forensic snapshot**.
3. Follow the steps in Upload a forensic snapshot to an AWS S3 bucket.

You can find the snapshots you generated in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Forensic Snapshots\`.

Snapshots generated from detections are in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Saved Data\`.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

Forensic snapshots

Forensic snapshots get data from a Sophos log of a computer's activity so that you can do your own analysis.

You can create a forensic snapshot from a threat graph or from the **Status** tab in a device's details page.

Go to **Global Settings** > **Forensic Snapshots**.

You can configure how much data you want in your snapshots and where you want to put them.

> **Note**
>
> The configuration options may not be available for all customers yet.

Set the time period for the forensic snapshot

By default, a snapshot includes data for the previous two weeks.

Here you can set a different time period or choose to include all the available data.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/GlobalSettings/ForensicSnapshots/index.html.)

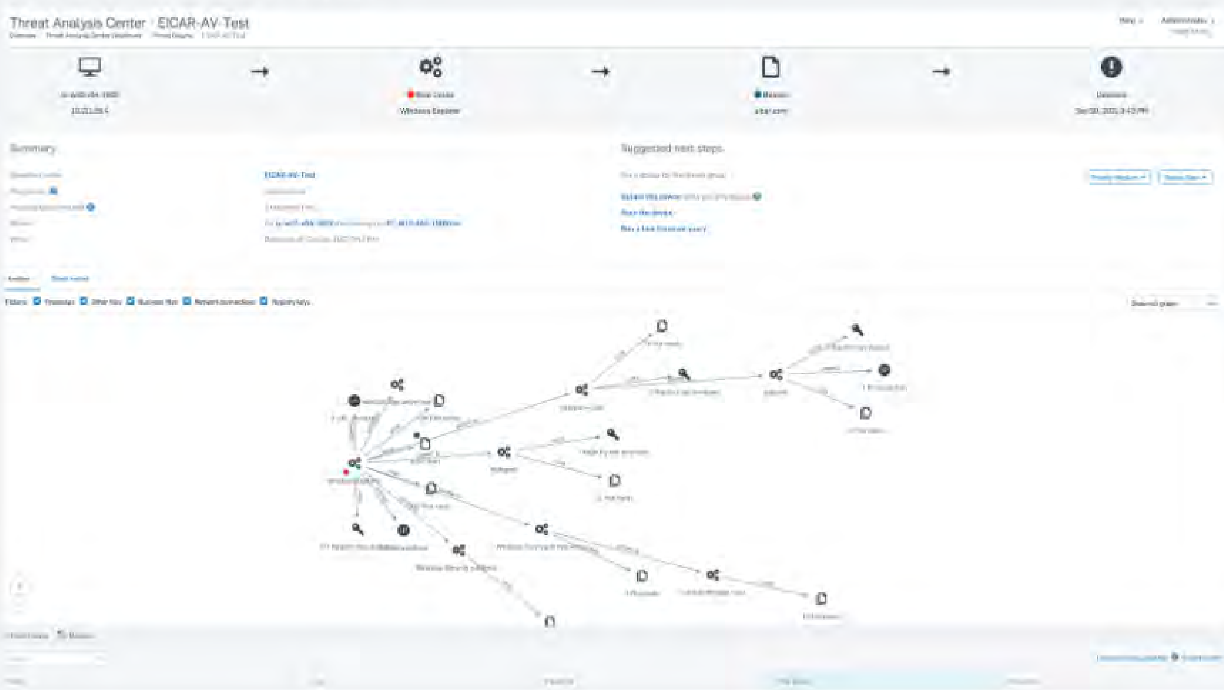
399. The Accused Products perform a method that includes *accessing, using a hardware processor of the computer, at least a portion of a recorded history of externally networked sources that the computer received files from so as to identify, based at least in part upon the identified indicia of pestware, a reference to an identity of an externally networked source that is suspected of originating pestware*. Intercept X's Threat Analysis Center dashboard includes "Threat Graphs" for investigating and cleaning up malware attacks. Users can "investigate a threat graph by going to its details pages and using the analysis tools." For example, the threat summary provides a summary of threat activity, including the root cause, "Where" the threat was detected (*e.g.* on which device), and "When" the threat was detected. The "Threat Graph analysis" also identifies a

reference to an identity of an externally networked source, such as "EICAR.com" in the example below, that is suspected of originating pestware. The threat graph analysis also shows the chain of events in the malware infection. Moreover, the threat graph analysis' "Graph record tab shows the history of the threat graph, from its creation by Sophos or the admin."

Threat Graph analysis

You can investigate a threat graph by going to its details page and using the analysis tools there.

Find the threat graph on the **Threat Graphs** page. Click its name to see a simplified event chain, summary, details of the artifacts (processes, files, keys) affected, and a diagram showing how the threat developed. The following screenshot shows an Eicar-AV-Test detection triggered by opening a file `eicar.com` in Windows Explorer.



(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

Summary

The **Summary** shows you a summary of the threat, including these details:

- **Root cause**: Where the infection entered your system.
- **Possible data involved**: Files that might contain important data. Check them to see if data has been encrypted or stolen.
- **Where**: Name of the device and its user.
- **When**: Detection time and date.

Analyze

The **Analyze** tab shows the chain of events in the malware infection.

A menu on the right of the tab lets you choose how much detail you see:

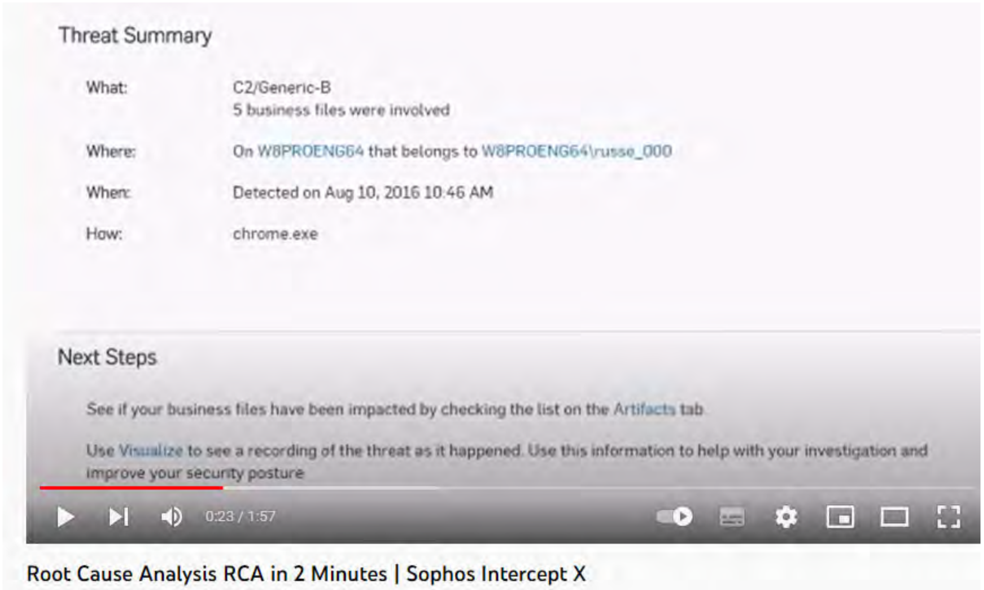
- **Show direct path**: This shows the chain of directly-involved items between the root cause and the item where the infection was detected (the "beacon").
- **Show full graph**: This shows the root cause, beacon, affected artifacts (applications, files, keys), the path of the infection (shown by arrows), and how the infection occurred. This is the default setting.

(*See*, https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

Graph record

The **Graph record** tab shows the history of the threat graph, from its creation by Sophos or the admin. You can post comments to record actions that have been taken and other relevant information.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)



(*See* https://www.youtube.com/watch?v=AOsjUjp4P7Q.)

400. In another example, Intercept X's Threat Analysis Center dashboard includes artifacts list to show "all affected items, for example business files, processes, registry keys, or IP addresses."

Artifacts list

This is a list below the diagram of the malware attack. It shows all the affected items, for example business files, processes, registry keys, or IP addresses.

You can export a comma separated (CSV) file containing a list of the affected artifacts, by clicking on **Export to CSV** at the top right of the tab.

The list shows:

- **Name**: Click the name to see more information in a details pane.
- **Type**: The type of artifact, such as a business file or a registry key.
- **Reputation**
- **Time logged**: The time and date a process was accessed.
- **Interactions**

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

401. Intercept X's Threat Analysis Center dashboard also generates a "forensic snapshot" from data it receives from the kernel-mode driver. The forensic snapshot is analyzed to identify the malicious activities.

Create forensic snapshot

You can create a "forensic snapshot" of data from the device. This gets data from a Sophos log of the device's activity and saves it on that device. For more information on forensic snapshots see Forensic snapshots.

You can also save it in the Amazon Web Services (AWS) S3 bucket you specify. You can then do your analysis.

You'll need a converter (which we provide) to read the data.

> **Note**
>
> You can choose how much data you want in snapshots and where to upload them. To do this, go to **Global Settings** > **Forensic Snapshots**. These options may not be available for all customers yet.

To create a snapshot, do as follows:

1. Go to a threat graph's **Analyze** tab.

   Alternatively, on the details page of the device, open the **Status** tab.
2. Click **Create forensic snapshot**.
3. Follow the steps in Upload a forensic snapshot to an AWS S3 bucket.

You can find the snapshots you generated in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Forensic Snapshots\`.

Snapshots generated from detections are in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Saved Data\`.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

Forensic snapshots

Forensic snapshots get data from a Sophos log of a computer's activity so that you can do your own analysis.

You can create a forensic snapshot from a threat graph or from the **Status** tab in a device's details page.

Go to **Global Settings** > **Forensic Snapshots**.

You can configure how much data you want in your snapshots and where you want to put them.

> **Note**
>
> The configuration options may not be available for all customers yet.

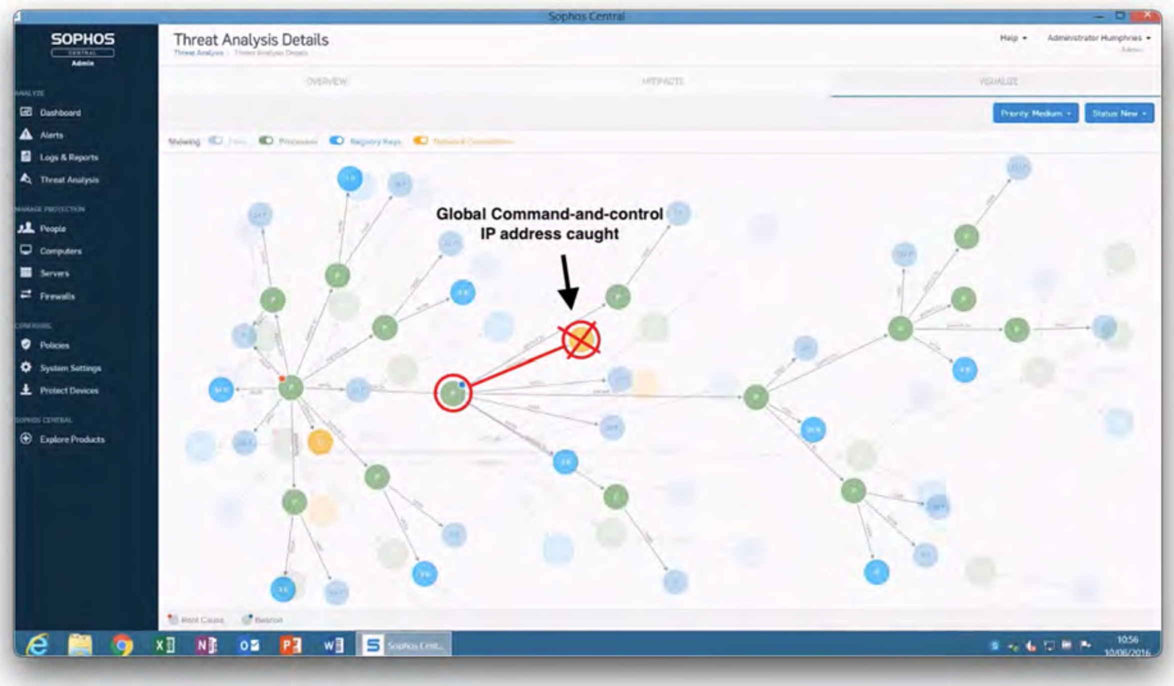
Set the time period for the forensic snapshot

By default, a snapshot includes data for the previous two weeks.

Here you can set a different time period or choose to include all the available data.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/GlobalSettings/ForensicSnapshots/index.html.)

402. Moreover, the Accused Products' root cause analyses identify, the IP addresses that originated each suspicious, or "beacon," event: "Here's the beacon event, the thing that was caught, in this case a fake salary report. Here's the root, Google Chrome. We see here that the fake report reached out to an IP address that we happen to know to be a command-and-control site. This right here is the moment of conviction, when we discovered that what was executing was a piece of malware."



(*See* https://youtu.be/AOsjUjp4P7Q?t=48.)

403. The Accused Products perform a method *wherein the recorded history of externally networked sources is stored on the file storage device*. For example, Intercept X's Threat Analysis Center dashboard generates a "forensic snapshot" and saves it in local file directory for future analysis. The forensic snapshot can also be saved in the Amazon Web Services (AWS) S3 bucket.

Create forensic snapshot

You can create a "forensic snapshot" of data from the device. This gets data from a Sophos log of the device's activity and saves it on that device. For more information on forensic snapshots see Forensic snapshots.

You can also save it in the Amazon Web Services (AWS) S3 bucket you specify. You can then do your analysis.

You'll need a converter (which we provide) to read the data.

> **Note**
>
> You can choose how much data you want in snapshots and where to upload them. To do this, go to **Global Settings** > **Forensic Snapshots**. These options may not be available for all customers yet.

To create a snapshot, do as follows:

1. Go to a threat graph's **Analyze** tab.

   Alternatively, on the details page of the device, open the **Status** tab.
2. Click **Create forensic snapshot**.
3. Follow the steps in Upload a forensic snapshot to an AWS S3 bucket.

You can find the snapshots you generated in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Forensic Snapshots\`.

Snapshots generated from detections are in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Saved Data\`.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

404. Each claim in the ’244 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ‘244 Patent.

405. Defendant directly infringes at least claim 1 of the ’244 Patent, either literally or under the doctrine of equivalents, by performing the steps described above. For example, Defendant performs the claimed method as described above by running the Sophos security software and corresponding systems to protect its own computer and network operations.

Defendant also performs the claimed method as described above when testing the operation of the Accused Products and corresponding systems. As another example, Defendant performs the claimed method as described above when providing or administering services to third parties, customers, and partners using the Accused Products.

406. Defendant's partners, customers, and end users of the Accused Products and corresponding systems and services directly infringe at least claim 1 of the '244 Patent, literally or under the doctrine of equivalents, at least by using the Accused Products and corresponding systems and services, as described above.

407. Defendant has actively induced and is actively inducing infringement of at least claim 1 of the '244 Patent with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b). For example, Sophos encourages and induces customers with specific intent to use Sophos's security software in a manner that infringes claim 1 of the '244 Patent at least by offering and providing software that performs a method that infringes claim 1 when installed and operated by the customer, and by engaging in activities relating to selling, marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including the activities described below.

408. Defendant encourages, instructs, directs, and/or requires third parties—including its certified partners and/or customers to perform the claimed method using the software, services, and systems in infringing ways, as described above.

409. Defendant further encourages and induces its customers to infringe claim 1 of the '244 Patent: 1) by making its security services available on its website, providing applications that allow users to access those services, widely advertising those services, and providing technical support and instructions to users, and 2) through activities relating to marketing, advertising,

promotion, installation, support, and distribution of the Accused Products, including its Sophos security software, and services in the United States. (*See* https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx; WBR_SOP000525, https://partners.sophos.com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1; WBR_SOP000209, https://secure2.sophos.com/en-us/security-news-trends/whitepapers/gated-wp/cybersecurity-system-buyers-guide.aspx%23formFrame.)

410. For example, Defendant shares instructions, guides, and manuals, which advertise and instruct third parties on how to use the software as described above, including at least customers and partners. (*Id*.) On further information and belief, Defendant provides customer service and technical support to purchasers of the Accused Products and corresponding system and services, which directs and encourages customers to perform certain actions that use the Accused Products in an infringing manner. (*Id*.)

411. Defendant and/or its partners recommend and sell the Accused Products and provide technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. Each customer enters into a contractual relationship with Defendant and/or one of its partners, which obligates each customer to perform certain actions in order to use the Accused Products. Further, in order to receive the benefit of Defendant's or its partner's continued technical support and their specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that performs the claimed method and infringes the '244 Patent.

412. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Product remains operational for each customer through ongoing technical support, Defendant and/or Sophos's partners affirmatively aid

and abet each customer's use of the Accused Products in a manner that performs the claimed method of, and infringes, the '244 Patent.

413. Defendant also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the claimed methods, and not a staple article or commodity of commerce suitable for substantial non-infringing uses. Indeed, as shown above, the Accused Products and the example functionality described below have no substantial non-infringing uses but are specifically designed to practice the '244 Patent.

414. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the functionality for identifying an origin of a malicious activity and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. For example, without generating a forensic snapshot, the Accused Products could not determine the origin of suspected pestware or malware. These processes are continually running when the system is in use and cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the '244 Patent, that functionality could not be performed.

415. Additionally, the accused functionality, including the functionality for identifying an origin of a malicious activity and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a

necessary part of that functionality. These processes are continually running when the system is in use and cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the '244 Patent, that functionality could not be performed.

416. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Defendant constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionality constitute a material part of the inventions claimed at least because such an attack chain is integral to the processes identified above (such as “monitoring, with a kernel-mode driver, activity on the computer”; “generating an activity log on a file storage device…”; “issuing a timestamp…”; “identifying, based upon the time of interest, indicia of pestware…”) as recited in the claims of the ’244 Patent. None of these products are staple goods—they are sophisticated and customized cyber security and malware detection products, methods, and systems.

417. Defendant’s infringing actions have continued after the original Complaint was filed. Defendant had knowledge of the ’244 Patent and of the specific acts that constitutes infringement of the ’244 Patent at least based on the filing of this amended Complaint, but has continued to engage in the infringing activity, including by selling, making, configuring, and installing the Accused Products and performing the accused functionality, and by engaging in activities that constitute infringement and contributory infringement as described above.

418. On information and belief, the infringing actions of each partner, customer, and/or end-user of the Accused Products are attributable to Defendant. For example, on information and

belief, Defendant directs and controls the activities or actions of its partners or others in connection with the Accused Products by contractual agreement or otherwise requiring partners or others to provide information and instructions to customers who acquire the Accused Products which, when followed, results in infringement. Defendant further directs and controls the operation of devices executing the Accused Products by programming the software which, when executed by a customer or end user, performs the method steps of at least claim 1 of the ’244 Patent.

419. Plaintiffs have suffered and continue to suffer damages, including lost profits, as a result of Defendant’s infringement of the ’244 Patent. Defendant is therefore liable to Plaintiffs under 35 U.S.C. § 284 for damages in an amount that adequately compensates Plaintiffs for Defendant’s infringement, but no less than a reasonable royalty.

420. Plaintiffs will continue to suffer irreparable harm unless this Court preliminarily and permanently enjoins Defendant, its agents, employees, representatives, and all others acting in concert with Defendant from infringing the ’244 Patent.

421. Defendant’s infringement of the ’244 Patent is knowing and willful. Defendant had actual knowledge of the ’244 Patent at least by the time Plaintiffs provided Defendant’s with a copy of this amended complaint, and had constructive knowledge of the ’244 Patent from at least the date Plaintiffs marked their products with the ’244 Patent and/or provided notice of the ’244 Patent on their website.

422. On information and belief, despite Defendant’s knowledge of the Asserted Patents and Plaintiffs’ patented technology, Defendant made the deliberate decision to sell products and services that they knew infringe the ’244 Patent. Defendant’s continued infringement of the ’244 Patent with knowledge of the ’244 Patent constitutes willful infringement.

TENTH CAUSE OF ACTION
(INFRINGEMENT OF THE ’243 PATENT)

423. Plaintiffs reallege and incorporate by reference the allegations of the preceding paragraphs of this Complaint.

424. Defendant has infringed and continues to infringe one or more claims of the ’243 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including Intercept X, at least when used for their ordinary and customary purposes, practice each element of at least claim 1 of the ’243 Patent, as described below.

425. For example, claim 1 of the ’243 Patent recites:

> 1. A method for identifying an origin of activity on a computer that is indicative of pestware comprising:
>
> monitoring, using a kernel-mode driver, the computer for activity that is indicative of pestware, wherein the monitoring includes monitoring API calls and storing a history of at least a portion of the API calls in an activity log;
>
> analyzing, heuristically, computer activity to determine whether one or more weighted factors associated with an activity exceeds a threshold so as to arrive at a determination that the activity is indicative of pestware;
>
> identifying, based upon the activity, an object residing on the computer that is a suspected pestware object;
>
> accessing, in response to the identifying an object, at least a portion of a recorded history of externally networked sources that the computer received files from so as to identify a reference to an identity of a particular externally networked source that the suspected pestware object originated from; and
>
> reporting the identity of the particular externally networked source to an externally networked pestware research entity so as to enable the externally networked pestware research entity to research whether the particular externally networked source is a source of pestware.

426. The Accused Products perform each element of the method of claim 1 of the ’243 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform *a method for identifying an origin of activity on a computer that is indicative of pestware*, as further

explained below. For example, Intercept X generates a root cause analysis to determine how the attacking process started and what else may have been happening on the device that is related to the root cause or detected escalation. The root cause analysis report includes detailed information of malicious activity and identifies how the attack unfolded and where it came from.

What happens when an attack is detected?

We will terminate the process and notify the end user. This will also initiate Sophos Clean to remove the malware. Upon detection, a root cause analysis will be generated to determine how the attacking process started and what else may have been happening on the device that is related to the root cause or detected escalation. The endpoint will be put into a red security health state, as this attack indicates an adversary has likely penetrated the device and more investigation is recommended.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

What should an admin do?

Like similar exploit prevention detections, administrators should review the root cause analysis report to determine how the attack unfolded and where it came from. Once the investigation is complete the administrator can clear the alert to allow normal operation of the device.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

Endpoint Detection and Response (EDR)

When malicious activity is detected on a device it is critical that the administrator has more information so that they can understand the nature of the incident, how it happened, what lead to the detection, and what actions that should be taken to prevent similar incidents in the future. This ability to perform forensics on an attack has been the work of security operations centers armed with SIEM and device forensic tools. Unfortunately, the volume of malicious activity detections in a typical organization can easily overwhelm understaffed IT security administrators. To address this challenge an expanded version of Intercept X, Intercept X Advanced with EDR, is available.

Top RCA questions

Is EDR available as a stand-alone product?

No, EDR capabilities are only included in Intercept X Advanced with EDR. This product combines all the capabilities of Intercept X and Central Endpoint Protection with EDR functionality.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-

Intercept-X-Solution-Brief.pdf.)

Malware expertise: Most organization rely on malware experts that specialize in reverse engineering to analyze suspicious files. Not only is this approach time consuming and difficult to achieve, but it assumes a level of cybersecurity sophistication which most organizations don't possess. Intercept X Advanced with EDR offers a better approach by leveraging Deep Learning Malware Analysis which automatically analyzes malware in extreme detail, breaking down file attributes and code and comparing them to millions of other files. Analysts can easily see which attributes and code segments are similar to "known-good" and "known bad" files so they can determine if a file should be blocked or allowed.

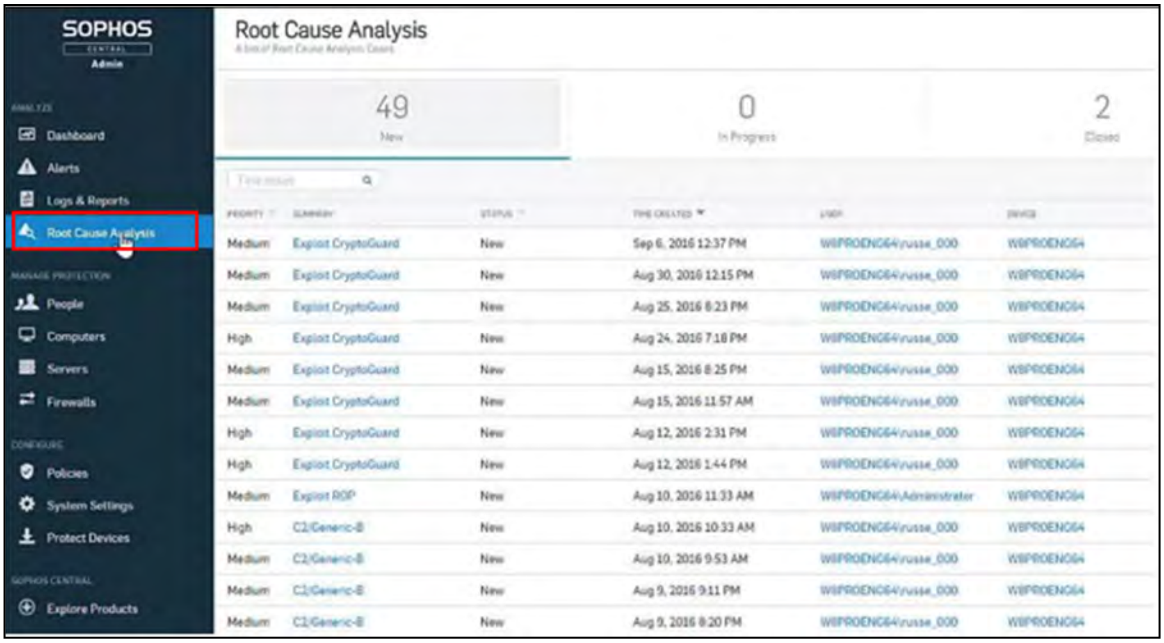
(*See* https://www.firewalls.com/pub/media/wysiwyg/datasheets/Sophos/intercept-x-edr.pdf.)

	Sophos Intercept X Advanced with EDR	Sophos Intercept X Advanced	Sophos Intercept X	Sophos Endpoint Protection
Foundational techniques	✓	✓		✓
Deep learning	✓	✓	✓	
Anti-exploit	✓	✓	✓	
CryptoGuard anti-ransomware	✓	✓	✓	
Endpoint detection and response (EDR)	✓			

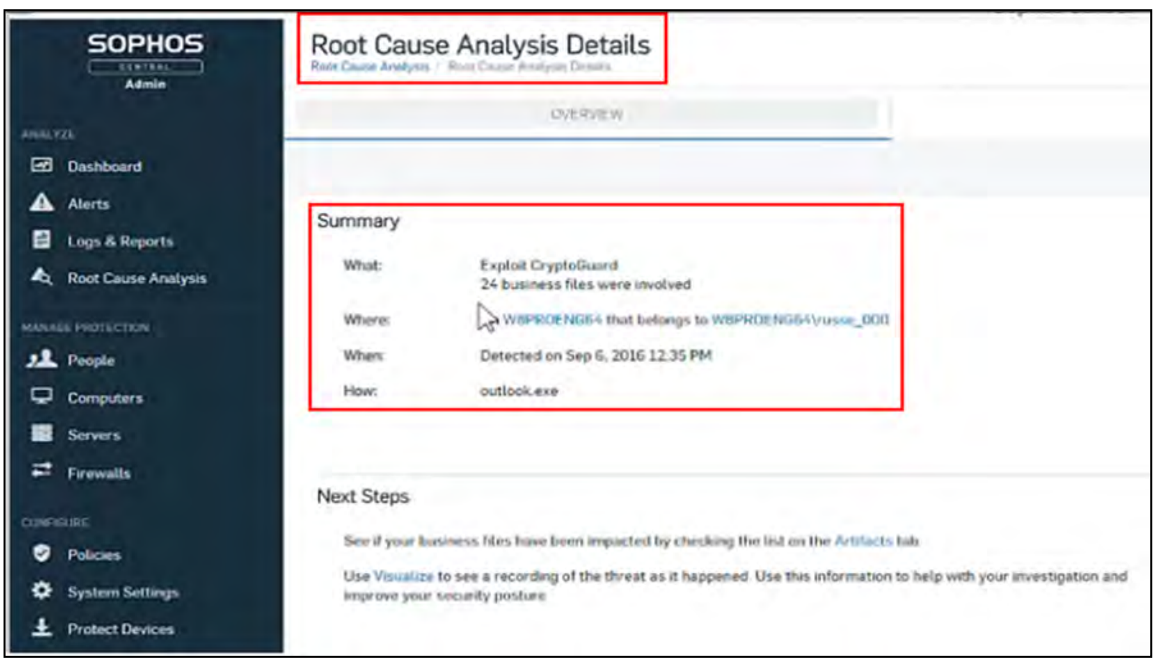
(*See* https://www.firewalls.com/pub/media/wysiwyg/datasheets/Sophos/intercept-x-edr.pdf.)

Sophos Intercept X Sophos Intercept X Advanced with EDR Sophos Intercept X for Server	Get the root cause analysis of an attack with complete visibility on the how and where of the attack along with recommendations on what your next steps should be.

(*See* https://assets.sophos.com/X24WTUEQ/at/pcxhcspjp7w9f79m4xqg4zq3/sophos-nist-compliance-card.pdf.)



(*See* https://www.sophos.com/en-us/medialibrary/Files/ppt/Solution%20partner%20event/SolPartner_Intercept X.pdf?la=en.pptx.)



(*See* https://www.sophos.com/en-us/medialibrary/Files/ppt/Solution%20partner%20event/SolPartner_Intercept X.pdf?la=en.pptx.)

Sophos Central Intercept X Advanced combines endpoint protection/detection and response into a single solution. It combines AI, behavioral analysis and tight exploit protection into a deep defense that can thwart most malware attacks. Let's break down Intercept X Advanced with EDR and see what sets it apart from the competition.

(*See* https://www.corporatearmor.com/product/sophos-central-intercept-x-advanced-with-edr-for-25-49-users-3-year-must-purchase-a-minimum-qty-of-25/.)

427. In addition, one of the core components of the root cause analysis is "threat case" assessment. Detection of malicious activity, including malware, exploits, ransomware detections, and blocking activity will trigger Intercept X's threat case algorithm to examine the contents of the data recorder. The objective of the algorithm is to take an identified malicious action, called a "beacon event," and then track events associated with the beacon back to an origin or root cause. The root cause can, for example, be the opening of an email attachment, the insertion of a USB drive, browsing to a website, and other similar activity. Once the root cause has been detected, the algorithm then moves forward from that event and identifies associated activity.

Threat cases – Detection of malicious activity, including malware, exploits, ransomware detections, and blocking activity will trigger the threat case algorithm to examine the contents of the data recorder. The objective of the algorithm is to take an identified malicious action, called a beacon event, and then track events associated with the beacon back to an origin or root cause. The root cause can be things like opening an email attachment, inserting a USB drive, browsing to website, and other activity. Once the root cause has been detected, the algorithm then moves forward from that event and identifies associated activity. Analysts can choose to view the full threat case, or the direct path only. By automatically tracking to the root cause and collecting associated data, Sophos is able to perform much of the work that a manual forensic examination would need to do. This dramatically reduces the time and expertise required to understand the origin of malicious activity.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

428. The Accused Products perform a method that includes *monitoring, using a kernel-mode driver, the computer for activity that is indicative of pestware, wherein the monitoring includes monitoring API calls and storing a history of at least a portion of the API calls in an activity log*. Intercept X monitors classes of applications at the kernel level. This injection into the process tree allows close and continuous monitoring of activities engaged in by a given process, including memory access, disk access, network access, DLLs loaded, and other process interactions.

429. Furthermore, Intercept X uses a "data recorder" that tracks activity on each device. The data recorder collects information on processes, memory, network, disk, and registry changes and both records that information locally on the device and makes it available over the cloud for "threat case" generation when an incident is detected. Activity over the last 30 days on the device is stored by the data recorder.

> **How does Intercept X prevent vulnerabilities being?**
>
> Intercept X monitors classes of applications at the kernel level. This injection into the process allows close and continuous monitoring of activity in the process, including memory access, disk, network access, DLLs loaded, and other process interactions.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

> Sophos Intercept X includes a novel approach to prevent attackers from addressing sensitive kernel functions via API functions that are unprotected.

(*See* https://assets.sophos.com/X24WTUEQ/at/cpbtrb4w4mfxqf4bn5cx8fh/sophos-comprehensive-exploit-prevention-wp.pdf.)

Root cause analysis core components

Sophos data recorder – Intercept X includes a data recorder that tracks activity on the device. Information collected on process, memory, network, disk, and registry changes are recorded locally on the device and made available for threat case generation when an incident is detected. Activity over the last 30 days on the device is stored and consumes about 100MB of disk space.

Threat cases – Detection of malicious activity, including malware, exploits, ransomware detections, and blocking activity will trigger the threat case algorithm to examine the contents of the data recorder. The objective of the algorithm is to take an identified malicious action, called a beacon event, and then track events associated with the beacon back to an origin or root cause. The root cause can be things like opening an email attachment, inserting a USB drive, browsing to website, and other activity. Once the root cause has been detected, the algorithm then moves forward from that event and identifies associated activity. Analysts can choose to view the full threat case, or the direct path only. By automatically tracking to the root cause and collecting associated data, Sophos is able to perform much of the work that a manual forensic examination would need to do. This dramatically reduces the time and expertise required to understand the origin of malicious activity.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

430. Moreover, the Accused Products monitor "sensitive API functions," include "Memory Scanning" for "defense against in-memory malware" and monitor "API call[s] (*e.g.*, VitrualAlloc)."

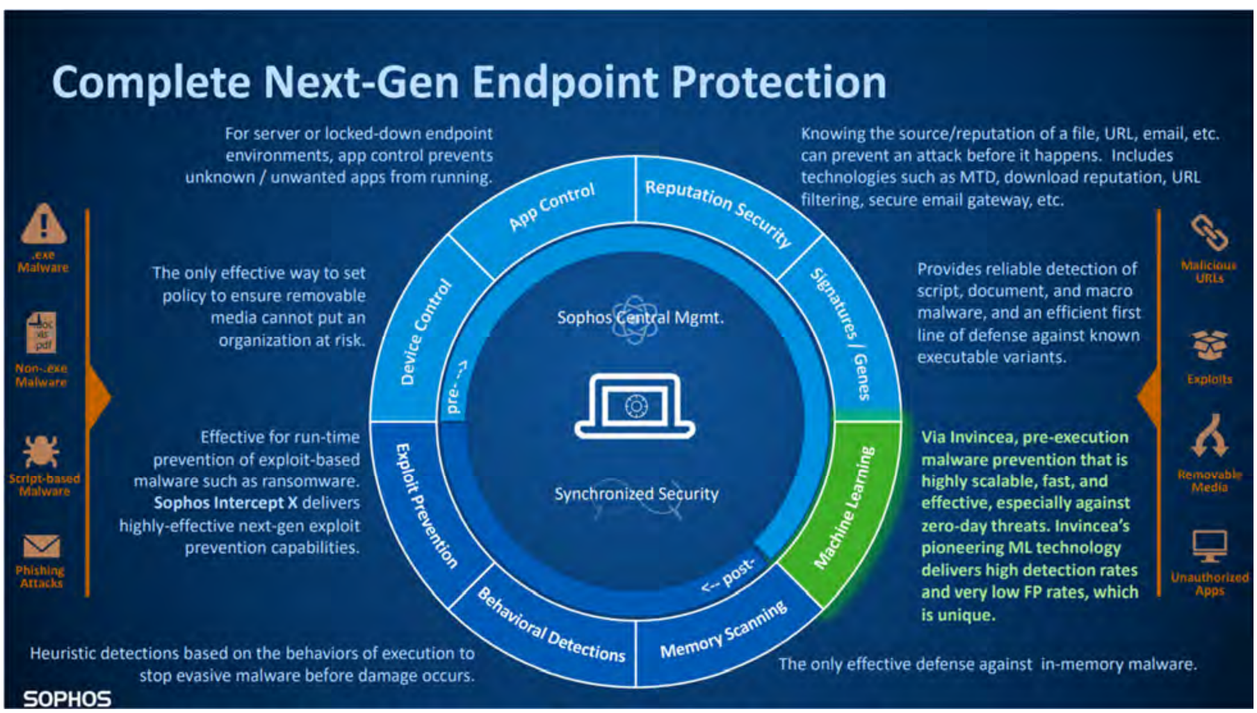
Stack-based ROP Mitigation (Caller)

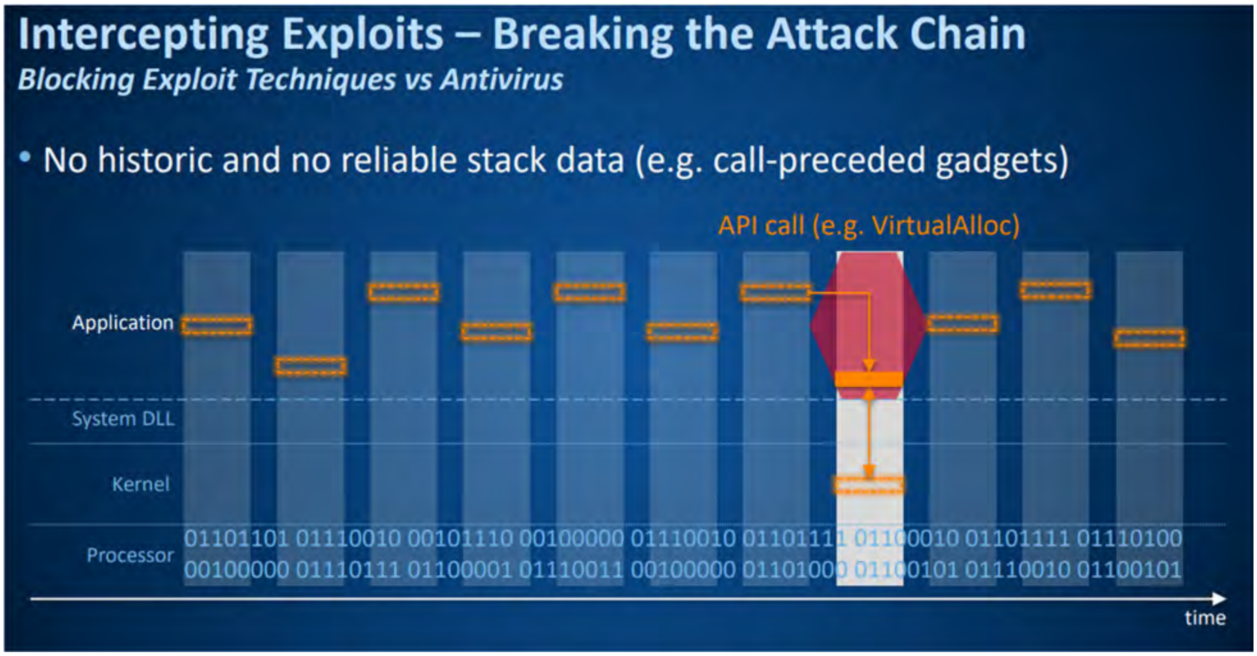
To defeat security technologies like data execution prevention (DEP) and address space layout randomization (ASLR), attackers typically resort to hijacking control-flow of vulnerable internet-facing applications. Such in-memory attacks are invisible to antivirus, most "next-gen" products, and other cyber defenses as there are no malicious files involved. Instead, the attack is constructed at run time by combining short pieces of benign code that are part of existing applications like Internet Explorer and Adobe Flash Player – a so-called code-reuse or return-oriented programming (ROP) attack.

During normal control-flow, sensitive API functions – like VirtualAlloc and CreateProcess – are invoked by the CALL instruction. Upon invoking a sensitive API, typical ROP defenses stop code execution to determine the API invoking address, using the 'return' address which is located on top of the stack. If the instruction of the API invoking address is not a CALL, the process is terminated.

Since the contents of the stack are writable, an attacker can write specific values on the stack to mislead the analysis of the stack-based ROP defense. The stack-based ROP defense cannot determine if the contents of the stack are benign or manipulated by an attacker.

(*See* WBR_SOP000387, https://www.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Comprehensive-Exploit-Prevention-wpna.pdf.)





(*See* WBR_SOP000566, https://secure2.sophos.com/it-it/medialibrary/PDFs/other/end-of-ransomware/MarkLomanSophosInterceptX.ashx.)

431. The Accused Products also evaluate and trace a stack "[u]pon invoking a sensitive API…to determine the API invoking address, using the 'return' address which is located on top of the stack." In another example, the Accused Products include the "CallerCheck" anti-exploit module for "[p]revent[ing] API invocation from stack memory." (*See* WBR_SOP000444, https://news.sophos.com/en-us/2021/03/04/covert-code-faces-a-heap-of-trouble-in-memory/.)

432. Furthermore, Intercept X monitors API calls to identify and detect malicious activities. For example, attackers can attempt to load malicious libraries by placing them on UNC paths. By monitoring calls to the LoadLibrary API, Intercept X prevents this type of illicit library loading.

Load Library

Attackers can attempt to load malicious libraries by placing them on UNC paths. Monitoring of all calls to the LoadLibrary API can be used to prevent this type of library loading.

(*See* https://assets.sophos.com/X24WTUEQ/at/cpbtrb4w4mfxqf4bn5cx8fh/sophos-comprehensive-exploit-prevention-wp.pdf.)

433. In addition, to preclude attackers from invoking the addresses of specific system functions through API calls (*e.g*. kernel32!VirtualProtect) via the "import address table" (IAT), Intercept X also filters out illicit calls by attackers to the IAT.

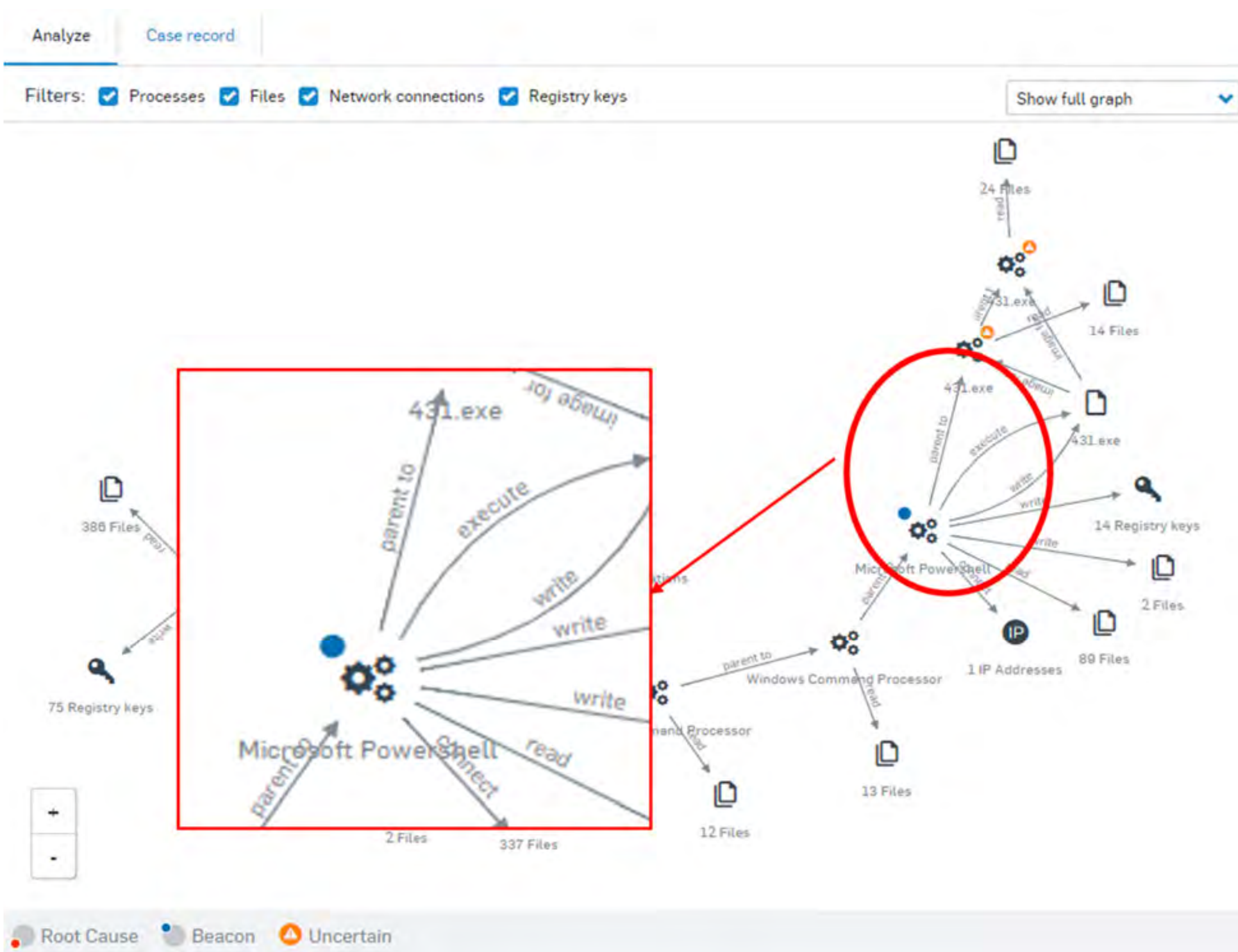
Import Address Table Access Filtering (IAF)

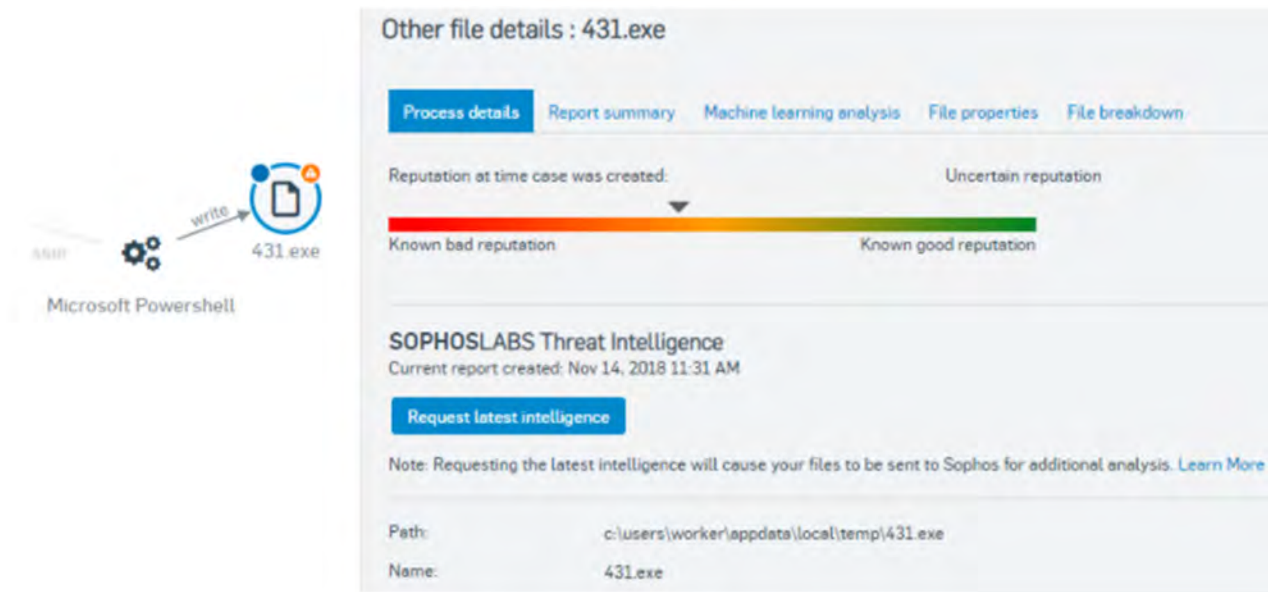
An attacker eventually needs the addresses of specific system functions (e.g. kernel32!VirtualProtect) to be able to perform malicious activities. These addresses can be retrieved from different sources, one of which is the import address table (IAT) of a loaded module. The IAT is used as a lookup table when an application calls a function in a different module. Because a compiled program cannot know the memory location of the libraries it depends upon, an indirect jump is required whenever an API call is made. As the dynamic linker loads modules and joins them together, it writes actual addresses into the IAT slots so that they point to the memory locations of the corresponding library functions.

(*See* https://assets.sophos.com/X24WTUEQ/at/cpbtrb4w4mfxqf4bn5cx8fh/sophos-comprehensive-exploit-prevention-wp.pdf.)

434. The Accused Products perform a method that includes *analyzing, heuristically, computer activity to determine whether one or more weighted factors associated with an activity exceeds a threshold so as to arrive at a determination that the activity is indicative of pestware and identifying, based upon the activity, an object residing on the computer that is a suspected pestware object*.

435. For example, the Accused Products assess "threat cases" by detecting illicit behaviors (*e.g.*, behaviors that are not normal for the object) such as the modification of "registry keys." In particular, in the example shown below, the "Analyze" tab of a "threat case" displayed in the "Threat Analytics Center" illustrates the illicit execution, on an infected endpoint device, of the suspicious process "431.exe" by Microsoft Powershell, which is marked as the "Beacon," and the subsequent modification of "registry keys" by the file "431.exe" on the infected endpoint. The process "431.exe" is quarantined after its behavior is detected as being malicious.





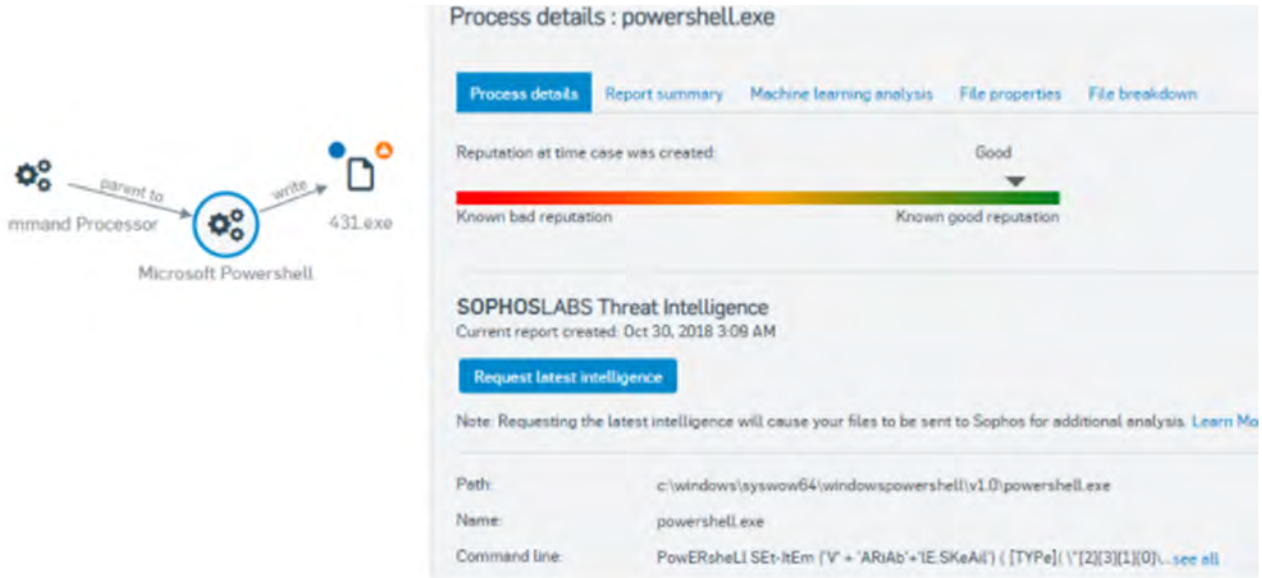
Important: Customers who are **not** using Sophos XDR, you will only see the one **Process details** tab.

For customers using Sophos EDR, by pressing the **Request latest intelligence** button, the file will be retrieved out of the Sophos quarantine and submitted to SophosLabs. A couple of minutes later the four other tabs (Report summary, Machine learning analysis, File properties, File breakdown) pictured will be displayed. The purpose of these these additional tabs is to help display the various properties of the file in a simple way. This can be useful for various reasons, one of them is to feel confident that the file is indeed malicious and not something you want in your environment. For more information on SophosLabs Threat Intelligence, please see: Sophos Central: Threat intelligence overview.

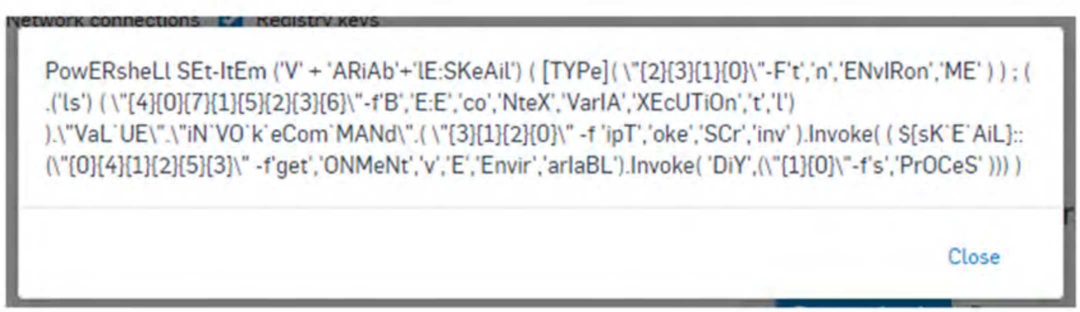
(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl.)

436. Based on the data that it received about the behavior of processes (*e.g.*, "431.exe" and the behavior of the process "Microsoft PowerShell," which executed "431.exe"), Intercept X determined that PowerShell exceeded the scope of normal behavior by being launched "with an obfuscated and very suspicious command line" and that "431.exe" illicitly modified registry keys and was written to "the users AppData location," which is "typically meant for data not executable," resulting in its quarantine. As such, Intercept X defines what is typical or normal behavior for both malware and non-malware. For example, it explains that "[o]bfuscation is very typical in malicious code and is designed to hide the true goal behind the code." In another example, when one command shell launches another command shell, that behavior is deemed

suspicious, meaning that when one command shell does not invoke another, it is deemed non-malware behavior: "We can also see that when CMD was launched it then launched another copy of CMD, this one with a similar suspicious command line."



We can also see that Powershell has been launched with an obfuscated and very suspicious command line.



(*See* WBR_SOP000264, https://support.sophos.com/support/s/article/KB-000036359?language =en_USl.)

437. Furthermore, the root cause analysis generated by Intercept X determines how the attacking process started and what else may have been happening on the device that is related to the root cause or detected escalation. The root cause analysis report includes detailed information of malicious activity and, for instance, identifies how the attack unfolded and where it came from.

What happens when an attack is detected?

We will terminate the process and notify the end user. This will also initiate Sophos Clean to remove the malware. Upon detection, a root cause analysis will be generated to determine how the attacking process started and what else may have been happening on the device that is related to the root cause or detected escalation. The endpoint will be put into a red security health state, as this attack indicates an adversary has likely penetrated the device and more investigation is recommended.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

What should an admin do?

Like similar exploit prevention detections, administrators should review the root cause analysis report to determine how the attack unfolded and where it came from. Once the investigation is complete the administrator can clear the alert to allow normal operation of the device.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

Endpoint Detection and Response (EDR)

When malicious activity is detected on a device it is critical that the administrator has more information so that they can understand the nature of the incident, how it happened, what lead to the detection, and what actions that should be taken to prevent similar incidents in the future. This ability to perform forensics on an attack has been the work of security operations centers armed with SIEM and device forensic tools. Unfortunately, the volume of malicious activity detections in a typical organization can easily overwhelm understaffed IT security administrators. To address this challenge an expanded version of Intercept X, Intercept X Advanced with EDR, is available.

Top RCA questions

Is EDR available as a stand-alone product?

No, EDR capabilities are only included in Intercept X Advanced with EDR. This product combines all the capabilities of Intercept X and Central Endpoint Protection with EDR functionality.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

Malware expertise: Most organization rely on malware experts that specialize in reverse engineering to analyze suspicious files. Not only is this approach time consuming and difficult to achieve, but it assumes a level of cybersecurity sophistication which most organizations don't possess. Intercept X Advanced with EDR offers a better approach by leveraging Deep Learning Malware Analysis which automatically analyzes malware in extreme detail, breaking down file attributes and code and comparing them to millions of other files. Analysts can easily see which attributes and code segments are similar to "known-good" and "known bad" files so they can determine if a file should be blocked or allowed.

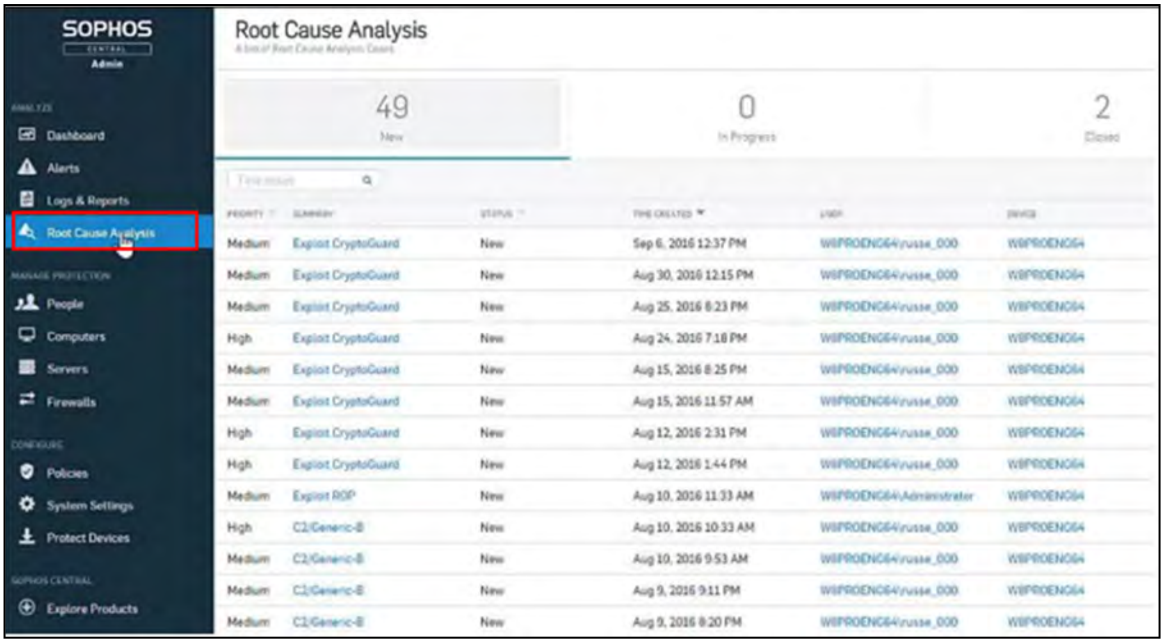
(*See* https://www.firewalls.com/pub/media/wysiwyg/datasheets/Sophos/intercept-x-edr.pdf.)

	Sophos Intercept X Advanced with EDR	Sophos Intercept X Advanced	Sophos Intercept X	Sophos Endpoint Protection
Foundational techniques	✓	✓		✓
Deep learning	✓	✓	✓	
Anti-exploit	✓	✓	✓	
CryptoGuard anti-ransomware	✓	✓	✓	
Endpoint detection and response (EDR)	✓			

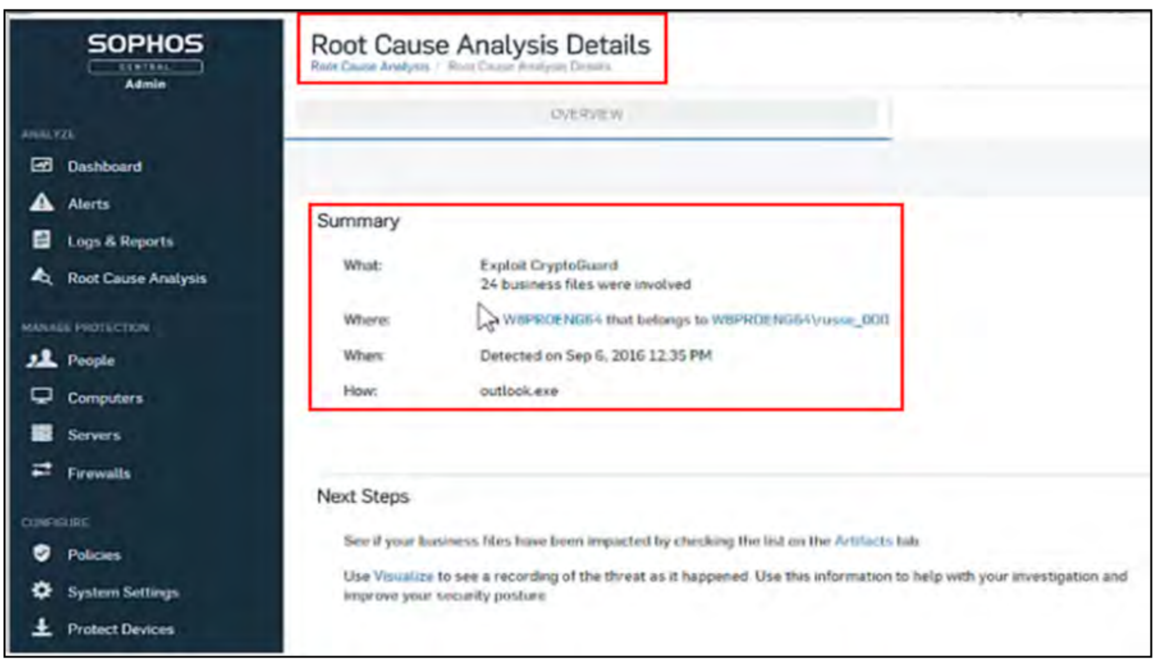
(*See* https://www.firewalls.com/pub/media/wysiwyg/datasheets/Sophos/intercept-x-edr.pdf.)

Sophos Intercept X Sophos Intercept X Advanced with EDR Sophos Intercept X for Server	Get the root cause analysis of an attack with complete visibility on the how and where of the attack along with recommendations on what your next steps should be.

(*See* https://assets.sophos.com/X24WTUEQ/at/pcxhcspjp7w9f79m4xqg4zq3/sophos-nist-compliance-card.pdf.)



(*See* https://www.sophos.com/en-us/medialibrary/Files/ppt/Solution%20partner%20event/ SolPartner_Intercept X.pdf?la=en.pptx.)



(*See* https://www.sophos.com/en-us/medialibrary/Files/ppt/Solution%20partner%20event/ SolPartner_Intercept X.pdf?la=en.pptx.)

> Sophos Central Intercept X Advanced combines endpoint protection/detection and response into a single solution. It combines AI, behavioral analysis and tight exploit protection into a deep defense that can thwart most malware attacks. Let's break down Intercept X Advanced with EDR and see what sets it apart from the competition.

(*See* https://www.corporatearmor.com/product/sophos-central-intercept-x-advanced-with-edr-for-25-49-users-3-year-must-purchase-a-minimum-qty-of-25/.)

438. Additionally, Intercept X's Threat Analysis Center assigns a "Risk" score to each activity it has detected as "unusual or suspicious" but has not yet blocked, and only displays such detections by default when they have a score equal to or greater than seven. Whether or not the detected threat is associated with a known "Mitre ATT&CK" technique increases its risk score. Such attack techniques include "Native API," whereby "[a]dversaries interact with native OS application programming interface[s] to execute behaviors."

Detections

Detections show you activity that you might need to investigate.

To see detections, go to **Threat Analysis Center** > **Detections**.

Detections identify activity on your devices that's unusual or suspicious but hasn't been blocked. They're different from events where we detect and block activity that we already know to be malicious.

We generate detections based on data that devices upload to the Sophos Data Lake.

We check that data against threat classification rules. When there's a match, we show a detection.

This page tells you how to use detections to look for potential threats.

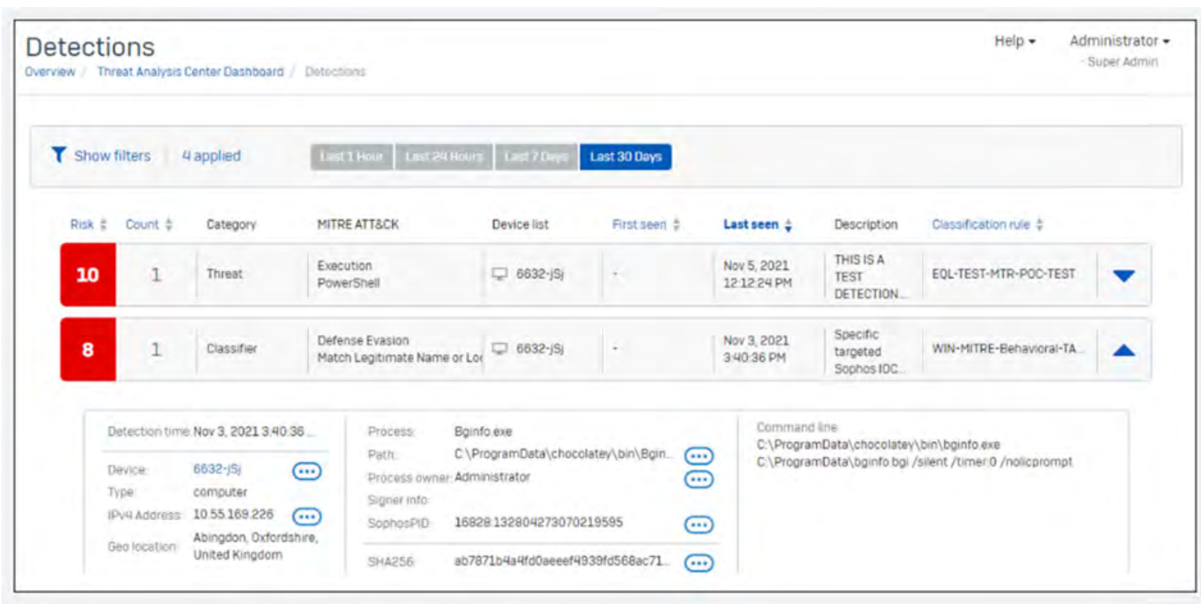
> **Note**
>
> **Investigations** can automatically group related detections together for more advanced analysis. See Investigations.

View detection details

To see detections, go to **Threat Analysis Center** > **Detections**.

We group detections according to the rule they matched and the date. The detection list shows the following:

- **Risk**. Risk is on a scale of 1 (lowest) to 10 (highest). With the default settings, we only show detections with a score of 7 or more. Use the score to prioritize investigations.
- **Classification rule**. The name of the rule that was matched.
- **Count**. Number of times the classification rule has been matched on a certain day.
- **Device list**. The device where the rule was last matched and the number of other devices with the same detection that day.
- **First seen** and **Last seen**. The first and last detections based on the classification rule that day.
- **Description**. What the rule identifies.
- **Mitre ATT&CK**. The corresponding Mitre ATT&CK Tactic and Technique.



(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/Detections/index.html.)

Native API

Adversaries may interact with the native OS application programming interface (API) to execute behaviors. Native APIs provide a controlled means of calling low-level OS services within the kernel, such as those involving hardware/devices, memory, and processes.[1][2] These native APIs are leveraged by the OS during system boot (when other system components are not yet initialized) as well as carrying out tasks and requests during routine operations.

(*See* https://attack.mitre.org/techniques/T1106/.)

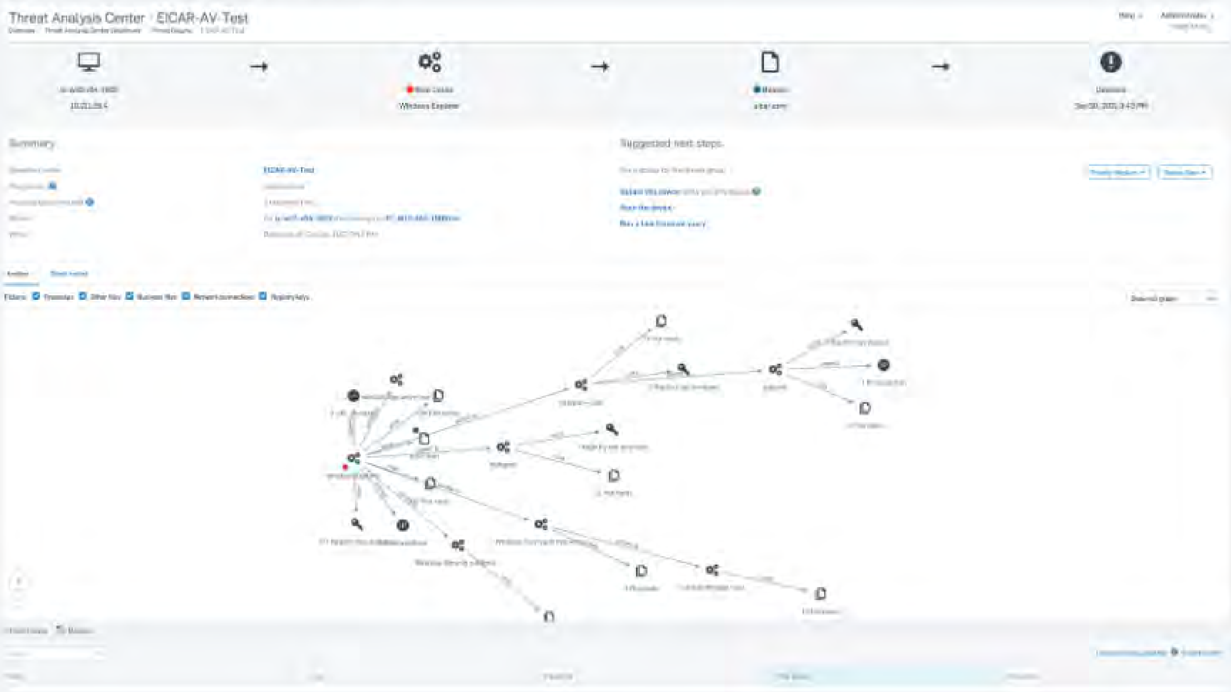
439. The Accused Products perform a method that includes "*accessing, in response to the identifying an object, at least a portion of a recorded history of externally networked sources that the computer received files from so as to identify a reference to an identity of a particular externally networked source that the suspected pestware object originated from*."

440. For example, Intercept X's Threat Analysis Center dashboard includes "Threat Graphs" for investigating and cleaning up malware attacks. Users can "investigate a threat graph by going to its details pages and using the analysis tools." For example, the threat summary provides a summary of threat activity, including the root cause, "Where" the threat was detected (*e.g.* on which device), and "When" the threat was detected. The "Threat Graph analysis" also identifies a reference to an identity of an externally networked source, such as "EICAR.com" in the example below, that is suspected of originating pestware. The threat graph analysis also shows the chain of events in the malware infection. Moreover, the threat graph analysis' "Graph record tab shows the history of the threat graph, from its creation by Sophos or the admin."

Threat Graph analysis

You can investigate a threat graph by going to its details page and using the analysis tools there.

Find the threat graph on the **Threat Graphs** page. Click its name to see a simplified event chain, summary, details of the artifacts (processes, files, keys) affected, and a diagram showing how the threat developed. The following screenshot shows an Eicar-AV-Test detection triggered by opening a file `eicar.com` in Windows Explorer.



(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

Summary

The **Summary** shows you a summary of the threat, including these details:

- **Root cause**: Where the infection entered your system.
- **Possible data involved**: Files that might contain important data. Check them to see if data has been encrypted or stolen.
- **Where**: Name of the device and its user.
- **When**: Detection time and date.

Analyze

The **Analyze** tab shows the chain of events in the malware infection.

A menu on the right of the tab lets you choose how much detail you see:

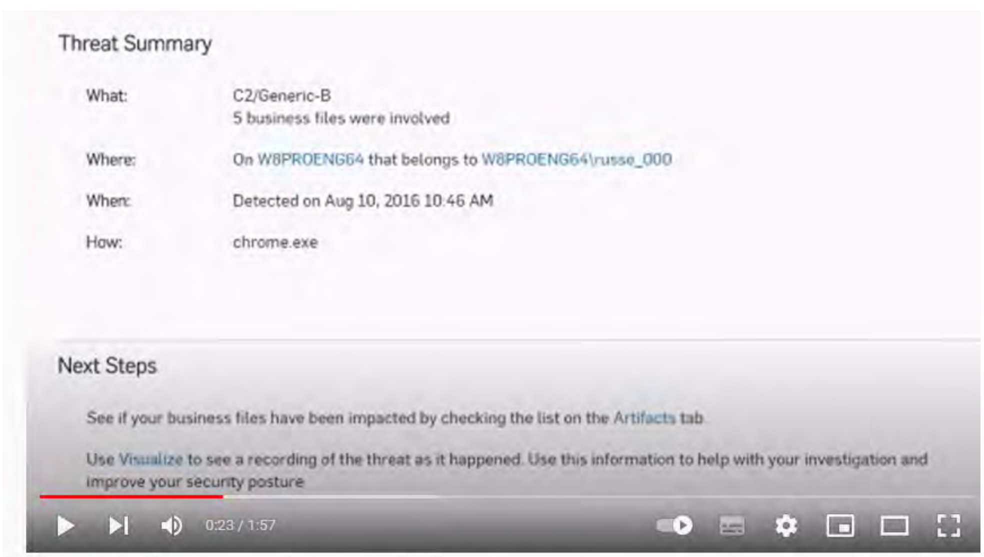
- **Show direct path**: This shows the chain of directly-involved items between the root cause and the item where the infection was detected (the "beacon").
- **Show full graph**: This shows the root cause, beacon, affected artifacts (applications, files, keys), the path of the infection (shown by arrows), and how the infection occurred. This is the default setting.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

Graph record

The **Graph record** tab shows the history of the threat graph, from its creation by Sophos or the admin. You can post comments to record actions that have been taken and other relevant information.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)



Root Cause Analysis RCA in 2 Minutes | Sophos Intercept X

(*See* https://www.youtube.com/watch?v=AOsjUjp4P7Q.)

441. In another example, Intercept X's Threat Analysis Center dashboard includes artifacts list to show "all affected items, for example business files, processes, registry keys, or IP addresses."

Artifacts list

This is a list below the diagram of the malware attack. It shows all the affected items, for example business files, processes, registry keys, or IP addresses.

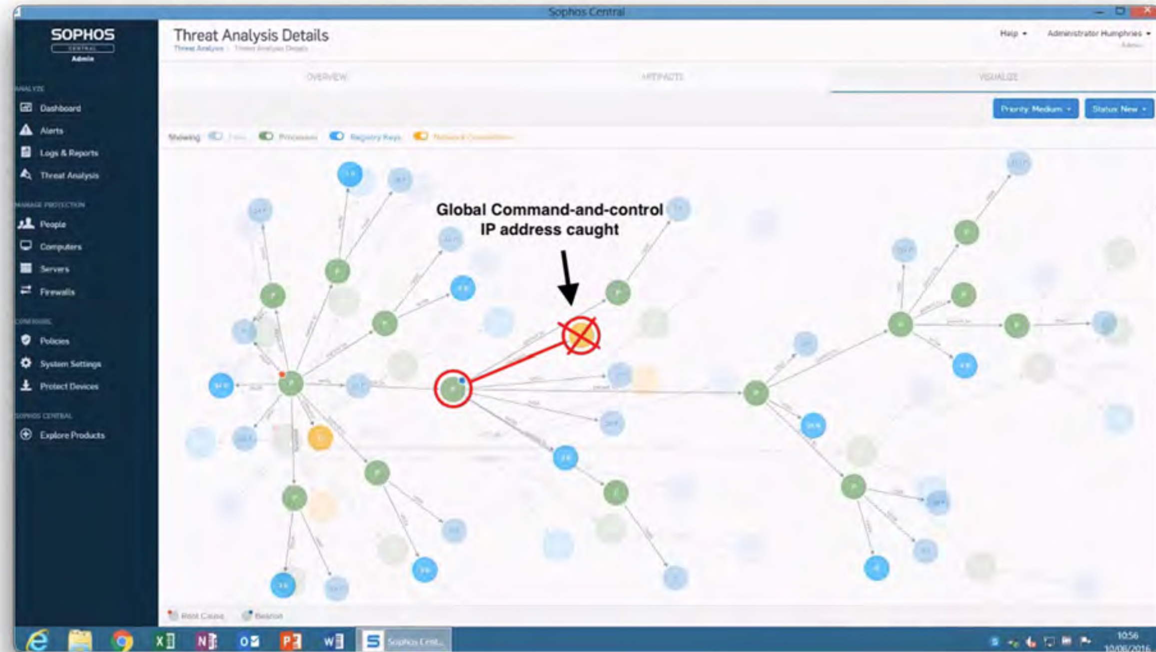
You can export a comma separated (CSV) file containing a list of the affected artifacts, by clicking on **Export to CSV** at the top right of the tab.

The list shows:

- **Name**: Click the name to see more information in a details pane.
- **Type**: The type of artifact, such as a business file or a registry key.
- **Reputation**
- **Time logged**: The time and date a process was accessed.
- **Interactions**

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

442. Moreover, Intercept X's root cause analyses identify, the IP addresses that originated each suspicious, or "beacon," event: "Here's the beacon event, the thing that was caught, in this case a fake salary report. Here's the root, Google Chrome. We see here that the fake report reached out to an IP address that we happen to know to be a command-and-control site. This right here is the moment of conviction, when we discovered that what was executing was a piece of malware."



(*See* https://youtu.be/AOsjUjp4P7Q?t=48.)

443. Intercept X's Threat Analysis Center dashboard also generates a "forensic snapshot" from data it receives from the kernel-mode driver. The forensic snapshot is analyzed to identify the malicious activities.

Create forensic snapshot

You can create a "forensic snapshot" of data from the device. This gets data from a Sophos log of the device's activity and saves it on that device. For more information on forensic snapshots see Forensic snapshots.

You can also save it in the Amazon Web Services (AWS) S3 bucket you specify. You can then do your analysis.

You'll need a converter (which we provide) to read the data.

> **Note**
>
> You can choose how much data you want in snapshots and where to upload them. To do this, go to **Global Settings** > **Forensic Snapshots**. These options may not be available for all customers yet.

To create a snapshot, do as follows:

1. Go to a threat graph's **Analyze** tab.

   Alternatively, on the details page of the device, open the **Status** tab.
2. Click **Create forensic snapshot**.
3. Follow the steps in Upload a forensic snapshot to an AWS S3 bucket.

You can find the snapshots you generated in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Forensic Snapshots\`.

Snapshots generated from detections are in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Saved Data\`.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

Forensic snapshots

Forensic snapshots get data from a Sophos log of a computer's activity so that you can do your own analysis.

You can create a forensic snapshot from a threat graph or from the **Status** tab in a device's details page.

Go to **Global Settings** > **Forensic Snapshots**.

You can configure how much data you want in your snapshots and where you want to put them.

> **Note**
>
> The configuration options may not be available for all customers yet.

Set the time period for the forensic snapshot

By default, a snapshot includes data for the previous two weeks.

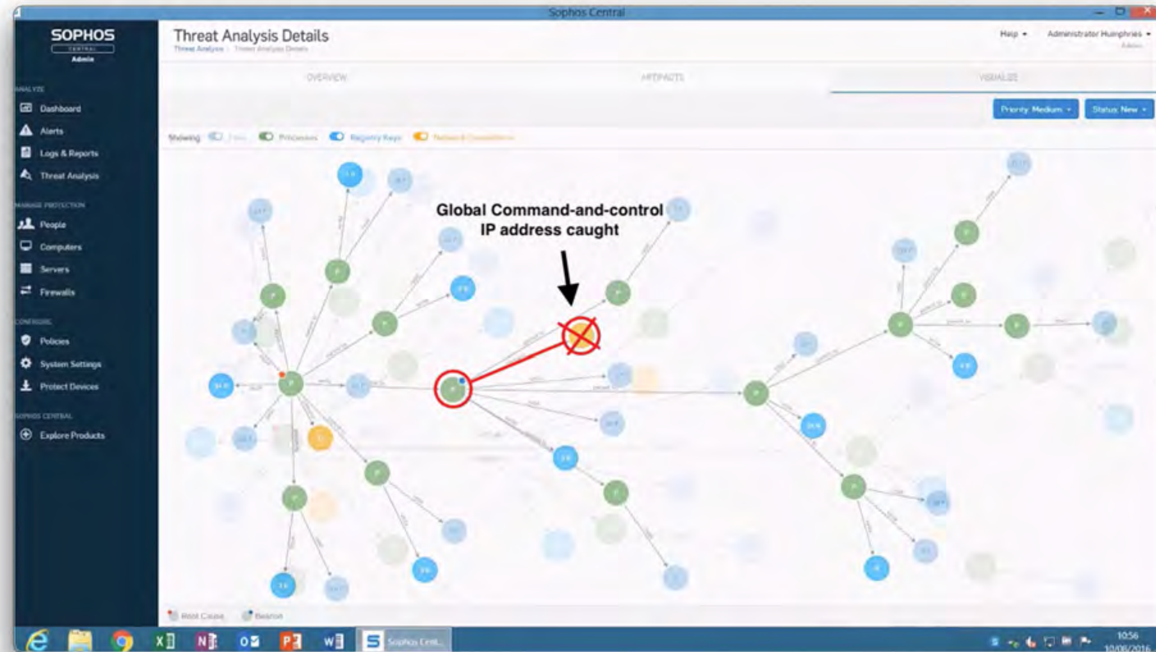
Here you can set a different time period or choose to include all the available data.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/GlobalSettings/ForensicSnapshots/index.html.)

444. The Accused Products perform a method that includes "*reporting the identity of the particular externally networked source to an externally networked pestware research entity so as to enable the externally networked pestware research entity to research whether the particular externally networked source is a source of pestware*."

445. For example, Intercept X's root cause analyses identify the IP addresses that originated each suspicious, or "beacon," event, and flag them as being "command-and-control" sites for malicious software if they are detected as being so by Sophos' threat analysis servers: "Here's the beacon event, the thing that was caught, in this case a fake salary report. Here's the root, Google Chrome. We see here that the fake report reached out to an IP address that we happen

to know to be a command-and-control site. This right here is the moment of conviction, when we discovered that what was executing was a piece of malware."



(*See* https://youtu.be/AOsjUjp4P7Q?t=48.)

446. Intercept X performs such root cause analyses and generates "detections based on data that users upload to the Sophos Data Lake."

Detections

Detections show you activity that you might need to investigate.

To see detections, go to **Threat Analysis Center** > **Detections**.

Detections identify activity on your devices that's unusual or suspicious but hasn't been blocked. They're different from events where we detect and block activity that we already know to be malicious.

We generate detections based on data that devices upload to the Sophos Data Lake.

We check that data against threat classification rules. When there's a match, we show a detection.

This page tells you how to use detections to look for potential threats.

Note

Investigations can automatically group related detections together for more advanced analysis. See Investigations.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/Detections/index.html.)

447. Each claim in the ’243 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ‘243 Patent.

448. Defendant directly infringes at least claim 1 of the ’243 Patent, either literally or under the doctrine of equivalents, by performing the steps described above. For example, Defendant performs the claimed method as described above by running the Sophos security software and corresponding systems to protect its own computer and network operations. Defendant also performs the claimed method as described above when testing the operation of the Accused Products and corresponding systems. As another example, Defendant performs the claimed method as described above when providing or administering services to third parties, customers, and partners using the Accused Products.

449. Defendant’s partners, customers, and end users of the Accused Products and

corresponding systems and services directly infringe at least claim 1 of the ’243 Patent, literally or under the doctrine of equivalents, at least by using the Accused Products and corresponding systems and services, as described above.

450. Defendant has actively induced and is actively inducing infringement of at least claim 1 of the ’243 Patent with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b). For example, Sophos encourages and induces customers with specific intent to use Sophos’s security software in a manner that infringes claim 1 of the ’243 Patent at least by offering and providing software that performs a method that infringes claim 1 when installed and operated by the customer, and by engaging in activities relating to selling, marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including the activities described below.

451. Defendant encourages, instructs, directs, and/or requires third parties—including its certified partners and/or customers to perform the claimed method using the software, services, and systems in infringing ways, as described above.

452. Defendant further encourages and induces its customers to infringe claim 1 of the ’243 Patent: 1) by making its security services available on its website, providing applications that allow users to access those services, widely advertising those services, and providing technical support and instructions to users, and 2) through activities relating to marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including its Sophos security software, and services in the United States. (*See* https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx; WBR_SOP000525, https://partners.sophos.com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1; WBR_SOP000209, https://secure2.sophos.com/en-us/security-news-trends/whitepapers/gated-

wp/cybersecurity-system-buyers-guide.aspx%23formFrame.)

453. For example, on information and belief, Defendant shares instructions, guides, and manuals, which advertise and instruct third parties on how to use the software as described above, including at least customers and partners. (*Id.*) On further information and belief, Defendant provides customer service and technical support to purchasers of the Accused Products and corresponding system and services, which directs and encourages customers to perform certain actions that use the Accused Products in an infringing manner. (*Id.*)

454. Defendant and/or its partners recommend and sell the Accused Products and provide technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Defendant and/or one of its partners, which obligates each customer to perform certain actions in order to use the Accused Products. Further, in order to receive the benefit of Defendant's or its partner's continued technical support and their specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that performs the claimed method and infringes the '243 Patent.

455. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Product remains operational for each customer through ongoing technical support, on information and belief, Defendant and/or Sophos's partners affirmatively aid and abet each customer's use of the Accused Products in a manner that performs the claimed method of, and infringes, the '243 Patent.

456. Defendant also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused

Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the claimed methods, and not a staple article or commodity of commerce suitable for substantial non-infringing uses. Indeed, as shown above, the Accused Products and the example functionality described below have no substantial non-infringing uses but are specifically designed to practice the ’243 Patent.

457. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including functionality for identifying an origin of a malicious activity, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’243 Patent, that functionality could not be performed.

458. Additionally, the accused functionality, including the functionality for identifying an origin of a malicious activity and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without deploying a kernel-mode driver that monitors processes for the API calls they make, and recording that activity, the Accused Products could not determine whether or not that object is pestware. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same

reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the '243 Patent, that functionality could not be performed.

459. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Defendant constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionality constitute a material part of the inventions claimed at least because such an attack chain is integral to the processes identified above (such as "monitoring, using a kernel-mode driver, the computer for activity…"; "analyzing, heuristically, computer activity to determine whether one or more weighted factors associated with an activity exceeds a threshold…"; "identifying, based upon the activity, an object residing on the computer…"; "reporting the identity of the particular externally networked source…") as recited in the claims of the '243 Patent. None of these products are staple goods—they are sophisticated and customized cyber security and malware detection products, methods, and systems.

460. Defendant's infringing actions have continued after the original Complaint was filed. Defendant had knowledge of the '243 Patent and of the specific acts that constitutes infringement of the '243 Patent at least on the date Defendant received a draft of this amended Complaint, but has continued to engage in the infringing activity, including by selling, making, configuring, and installing the Accused Products and performing the accused functionality, and by engaging in activities that constituted infringement and contributory infringement as described above.

461. On information and belief, the infringing actions of each partner, customer, and/or end-user of the Accused Products are attributable to Defendant. For example, on information and

belief, Defendant directs and controls the activities or actions of its partners or others in connection with the Accused Products by contractual agreement or otherwise requiring partners or others to provide information and instructions to customers who acquire the Accused Products which, when followed, results in infringement. Defendant further directs and controls the operation of devices executing the Accused Products by programming the software which, when executed by a customer or end user, perform the method steps of at least claim 1 of the ’243 Patent.

462. Plaintiffs have suffered and continue to suffer damages, including lost profits, as a result of Defendant’s infringement of the ’243 Patent. Defendant is therefore liable to Plaintiffs under 35 U.S.C. § 284 for damages in an amount that adequately compensates Plaintiffs for Defendant’s infringement, but no less than a reasonable royalty.

463. Plaintiffs will continue to suffer irreparable harm unless this Court preliminarily and permanently enjoins Defendant, its agents, employees, representatives, and all others acting in concert with Defendant from infringing the ’243 Patent.

464. Defendant’s infringement of the ’243 Patent is knowing and willful. Defendant had actual knowledge of the ’243 Patent at least by the time Defendant received a copy of this amended complaint, and had constructive knowledge of the ’243 Patent from at least the date Plaintiffs marked their products with the ’243 Patent and/or provided notice of the ’243 Patent on their website.

465. On information and belief, despite Defendant’s knowledge of the Asserted Patents and Plaintiffs’ patented technology, Defendant made the deliberate decision to sell products and services that they knew infringe the ’243 Patent. Defendant’s continued infringement of the ’243 Patent with knowledge of the ’243 Patent constitutes willful infringement.

ELEVENTH CAUSE OF ACTION
(INFRINGEMENT OF THE ’505 PATENT)

466. Plaintiffs reallege and incorporate by reference the allegations of the preceding paragraphs of this Complaint.

467. Defendant has infringed and continues to infringe one or more claims of the ’505 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including Intercept X, at least when used for their ordinary and customary purposes, practice each element of at least claim 1 of the ’505 Patent, as described below.

468. For example, claim 1 of the ’505 Patent recites:

> 1. A method for managing pestware on a computer comprising:
>
> monitoring events during a boot sequence of the computer;
>
> managing pestware-related events during a first period in a boot sequence of the computer, the first period in the boot sequence occurring before the computer becomes configured to run native applications, before a subsystem of an operating system is loaded, and after a kernel is loaded;
>
> managing pestware-related events in accordance with a set of behavior rules during a second period in the boot sequence occurring when the computer is configured to run native applications;
>
> generating, in response to the monitoring, a record of events, the record of events including the pestware-related events;
>
> analyzing the record of events so as to identify the pestware-related events;
>
> modifying the set of behavior rules so as to prevent the pestware related events; and
>
> scanning data in a registry of the computer for pestware during the second period in the boot sequence.

469. The Accused Products perform each element of the method of claim 1 of the ’505 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform *a method for managing pestware on a computer*, as further explained below. For example, Intercept

X offers malware detection, exploit protection, and other endpoint protection features to combat zero-day security threats.

Intelligent Endpoint Detection and Response
Sophos Intercept X Advanced with EDR integrates intelligent endpoint detection and response (EDR) with the industry's top-rated malware detection, top-rated exploit protection, and other unmatched endpoint protection features.

(*See* https://www.enterpriseav.com/datasheets/intercept-x-edr.pdf.)

EDR Starts with the Strongest Protection
To stop breaches before they start, prevention is crucial. Intercept X consolidates unmatched protection and endpoint detection and response into a single solution. This means that most threats are stopped before they can ever cause damage, and Intercept X Advanced with EDR provides additional cybersecurity assurance with the ability to detect, investigate, and respond to potential security threats.

(*See* https://www.enterpriseav.com/datasheets/intercept-x-edr.pdf.)

For example: Macros in documents are potentially dangerous, as they are created in the Visual Basic for Applications (VBA) programming language, which includes the ability to download and run binaries from the web and also allows the use of powershell and other trusted applications. This unexpected feature (or logic-flaw exploit) offers attackers an obvious advantage as they do not need to exploit a software bug or find a way to bypass code and memory defenses to infect computers. They simply abuse standard functionality offered by a widely-used trusted application and only need to use social engineering to persuade the victim to open the specially-crafted document.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

Endpoint Detection and Response (EDR)

When malicious activity is detected on a device it is critical that the administrator has more information so that they can understand the nature of the incident, how it happened, what lead to the detection, and what actions that should be taken to prevent similar incidents in the future. This ability to perform forensics on an attack has been the work of security operations centers armed with SIEM and device forensic tools. Unfortunately, the volume of malicious activity detections in a typical organization can easily overwhelm understaffed IT security administrators. To address this challenge an expanded version of Intercept X, **Intercept X Advanced with EDR**, is available.

Top RCA questions

Is EDR available as a stand-alone product?

No, EDR capabilities are only included in Intercept X Advanced with EDR. This product combines all the capabilities of Intercept X and Central Endpoint Protection with EDR functionality.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

470. The Accused Products perform a method that includes *monitoring events during a boot sequence of the computer and managing pestware-related events during a first period in a boot sequence of the computer, the first period in the boot sequence occurring before the computer becomes configured to run native applications, before a subsystem of an operating system is loaded, and after a kernel is loaded*. For example, Intercept X deploys a "Sophos Boot Driver" that executes when "malware cleanup is required on a computer reboot."

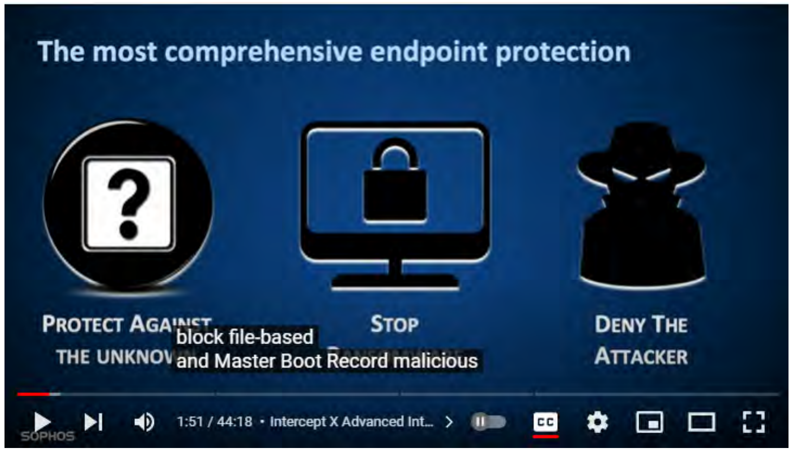
Sophos Boot Driver	SophosBootDriver	Not available	The driver will only be Running when malware cleanup is required on a computer reboot.

(*See* https://support.sophos.com/support/s/article/KB-000036446?language=en_US.)

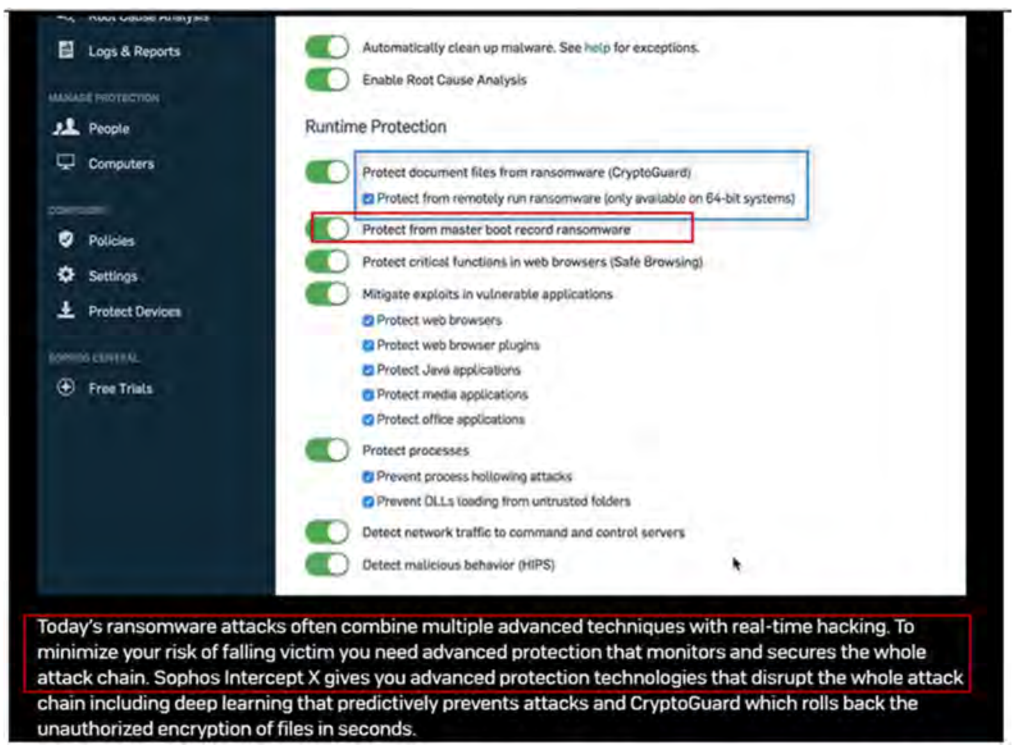
471. Working in conjunction with its boot driver, Intercept X deploys "Disk and Boot Record Protection (WipeGuard)" to combat ransomware that attempts to encrypt the Master Boot Record, which would prevent users from accessing their systems, and both detects and removes "rootkits" and "bootkits" "before the loading of a machine's operating system." Such bootkits and rootkits infiltrate software that runs during the boot process of a computer and target sensitive code such as the Master Boot Record, an early stage code executed when starting up a computer.

Features	Intercept X Advanced
Disk and Boot Record Protection (WipeGuard)	✓

(*See* https://www.sophos.com/en-us/products/endpoint-antivirus/tech-specs.)



(*See* https://www.youtube.com/watch?v=N7tNcGz4FVY.)



(*See* https://www.sophos.com/en-us/products/endpoint-antivirus.)

For example: Macros in documents are potentially dangerous, as they are created in the Visual Basic for Applications (VBA) programming language, which includes the ability to download and run binaries from the web and also allows the use of powershell and other trusted applications. This unexpected feature (or logic-flaw exploit) offers attackers an obvious advantage as they do not need to exploit a software bug or find a way to bypass code and memory defenses to infect computers. They simply abuse standard functionality offered by a widely-used trusted application and only need to use social engineering to persuade the victim to open the specially-crafted document.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

Sophos Rootkit and Bootkit Protection

Malware comes in many forms, all of them bad. Some of the worst among them are rootkits and bootkits. Rootkits can lie hidden on computers, remaining undetected by antivirus software. Although new rootkits can be prevented from infecting the system, any rootkits present before your antivirus was installed may never be revealed.

It helps to understand what these concepts mean for users. 'Rootkit' comes from the concept of root-level privileges on a device – administrator level, privileged access. If malware has that kind of control, everything is up for grabs. Rootkits are designed to grant the bad guys access they otherwise would not be allowed. Bootkits are an advanced form of rootkit. They go even further, seeking to infect the master boot record or volume boot record, so it can act even before the loading of the machine's operating system.

Rootkits are particularly hard to find once they're on your system. The whole point of rootkits is to hide malware, after all. First and foremost, a powerful, next-gen antivirus tool is a must-have. But there are other, more proactive steps you as the user can take to keep yourself safe.

(*See* https://home.sophos.com/en-us/content/bootkit-rootkit-removal.)

472. Working in conjunction with its Sophos Boot Driver, Intercept X can also detect and remedy events that are associated with "threat classification rules" and known "Mitre ATT&CK" techniques. Such rules include classification of illicit driver behavior as corresponding to the "Native API" technique, whereby "[a]dversaries interact with native OS application programming interface[s] to execute behaviors." Such "native APIs" are "leveraged by the OS during system boot."

Detections

Detections show you activity that you might need to investigate.

To see detections, go to **Threat Analysis Center** > **Detections**.

Detections identify activity on your devices that's unusual or suspicious but hasn't been blocked. They're different from events where we detect and block activity that we already know to be malicious.

We generate detections based on data that devices upload to the Sophos Data Lake.

We check that data against threat classification rules. When there's a match, we show a detection.

This page tells you how to use detections to look for potential threats.

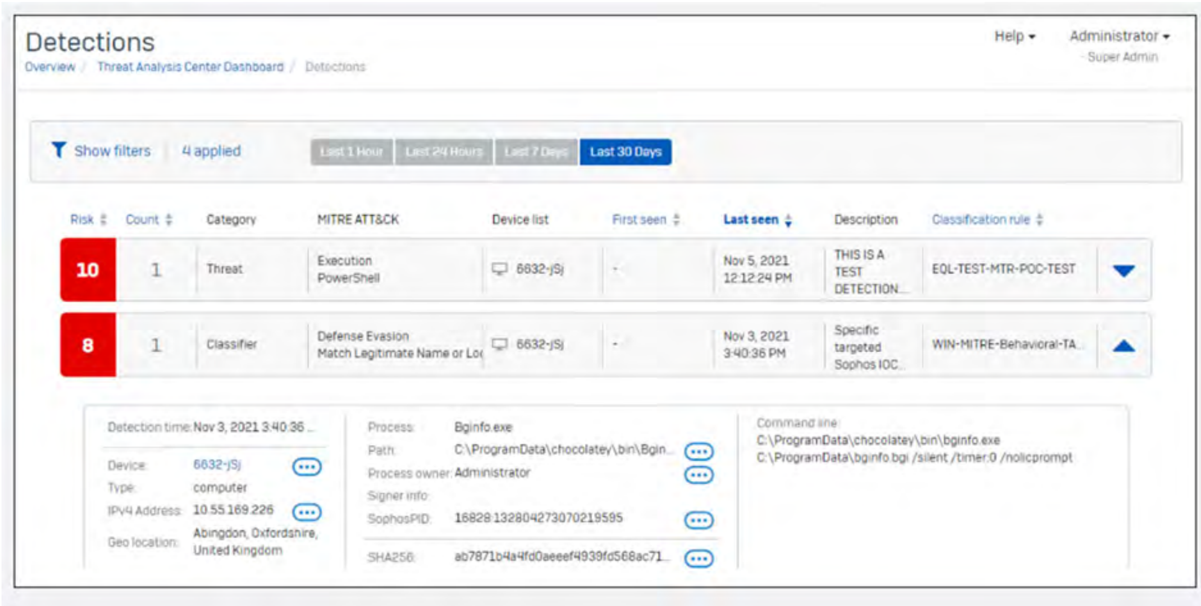
> **Note**
>
> **Investigations** can automatically group related detections together for more advanced analysis. See Investigations.

View detection details

To see detections, go to **Threat Analysis Center** > **Detections**.

We group detections according to the rule they matched and the date. The detection list shows the following:

- **Risk**. Risk is on a scale of 1 (lowest) to 10 (highest). With the default settings, we only show detections with a score of 7 or more. Use the score to prioritize investigations.
- **Classification rule**. The name of the rule that was matched.
- **Count**. Number of times the classification rule has been matched on a certain day.
- **Device list**. The device where the rule was last matched and the number of other devices with the same detection that day.
- **First seen** and **Last seen**. The first and last detections based on the classification rule that day.
- **Description**. What the rule identifies.
- **Mitre ATT&CK**. The corresponding Mitre ATT&CK Tactic and Technique.



(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/Detections/index.html.)

Native API

Adversaries may interact with the native OS application programming interface (API) to execute behaviors. Native APIs provide a controlled means of calling low-level OS services within the kernel, such as those involving hardware/devices, memory, and processes.[1][2] These native APIs are leveraged by the OS during system boot (when other system components are not yet initialized) as well as carrying out tasks and requests during routine operations.

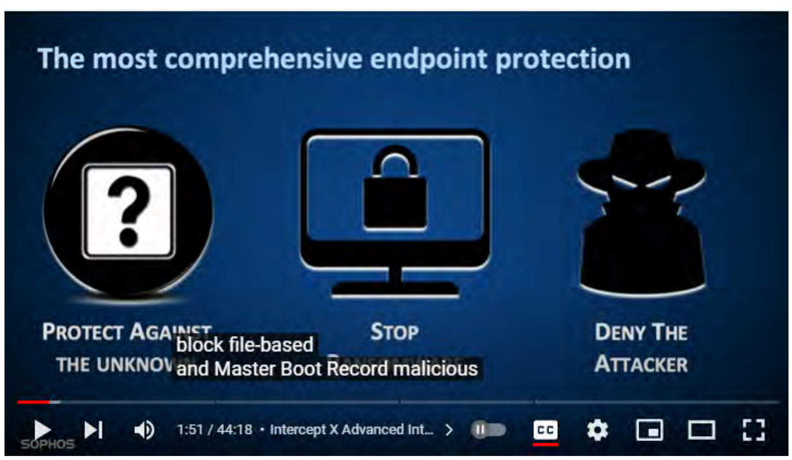
(*See* https://attack.mitre.org/techniques/T1106/.)

473. The Accused Products perform a method that includes *managing pestware-related events in accordance with a set of behavior rules during a second period in the boot sequence occurring when the computer is configured to run native applications*. For example, as the boot process continues, Intercept X employs “behavioral analysis to stop ransomware and boot record attacks” and includes “Disk and Boot Record Protection (WipeGuard).” In addition, Intercept X Advanced blocks Master Boot Record malicious encryption.

(*See* https://www.sophos.com/en-us/products/endpoint-antivirus.)

Features	Intercept X Advanced
Disk and Boot Record Protection (WipeGuard)	✓

(*See* https://www.sophos.com/en-us/products/endpoint-antivirus/tech-specs.)



(*See* https://www.youtube.com/watch?v=N7tNcGz4FVY.)

474. In addition, Intercept X protects against rootkits and bootkits that infiltrate software that runs during the boot process of a computer and target sensitive code such as the Master Boot Record, an early stage code executed when starting up a computer.

Sophos Rootkit and Bootkit Protection

Malware comes in many forms, all of them bad. Some of the worst among them are rootkits and bootkits. Rootkits can lie hidden on computers, remaining undetected by antivirus software. Although new rootkits can be prevented from infecting the system, any rootkits present before your antivirus was installed may never be revealed.

It helps to understand what these concepts mean for users. 'Rootkit' comes from the concept of root-level privileges on a device – administrator level, privileged access. If malware has that kind of control, everything is up for grabs. Rootkits are designed to grant the bad guys access they otherwise would not be allowed. Bootkits are an advanced form of rootkit. They go even further, seeking to infect the master boot record or volume boot record, so it can act even before the loading of the machine's operating system.

Rootkits are particularly hard to find once they're on your system. The whole point of rootkits is to hide malware, after all. First and foremost, a powerful, next-gen antivirus tool is a must-have. But there are other, more proactive steps you as the user can take to keep yourself safe.

(*See* https://home.sophos.com/en-us/content/bootkit-rootkit-removal.)

475. Moreover, Intercept X monitors "sensitive API functions" and includes "Memory Scanning" for "defense against in-memory malware" and monitor "API call[s] (*e.g.*, VitrualAlloc)."

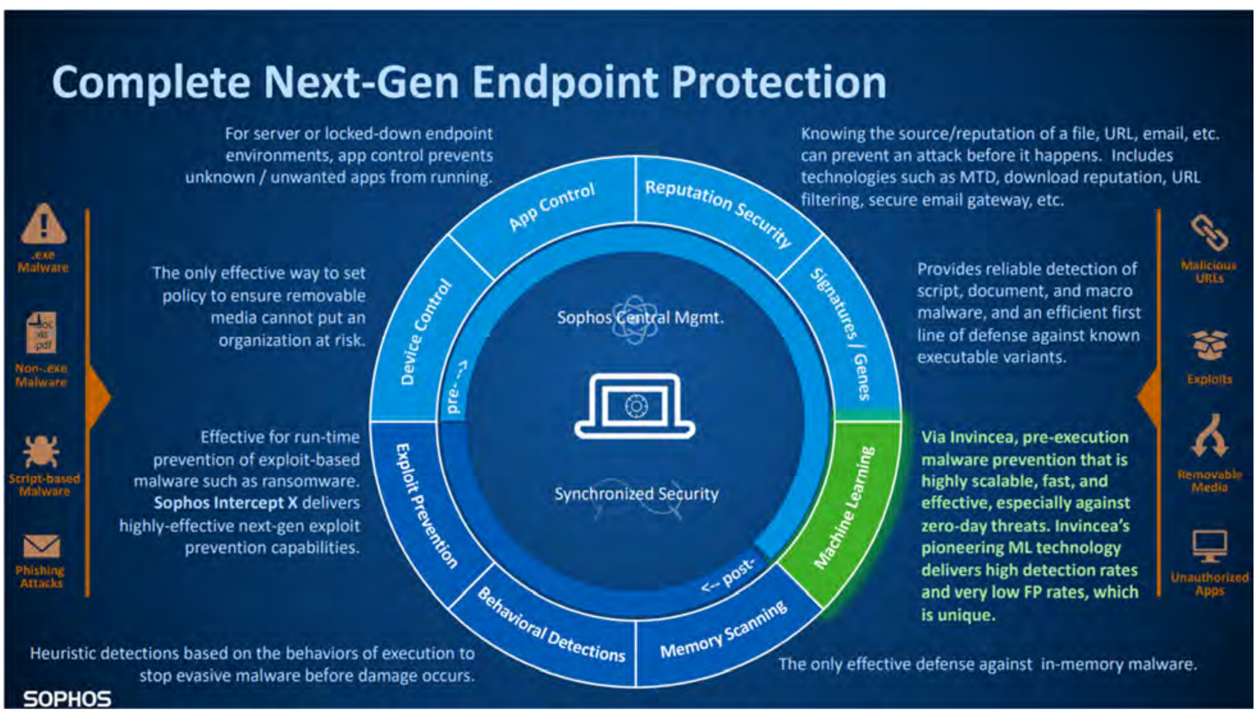
Stack-based ROP Mitigation (Caller)

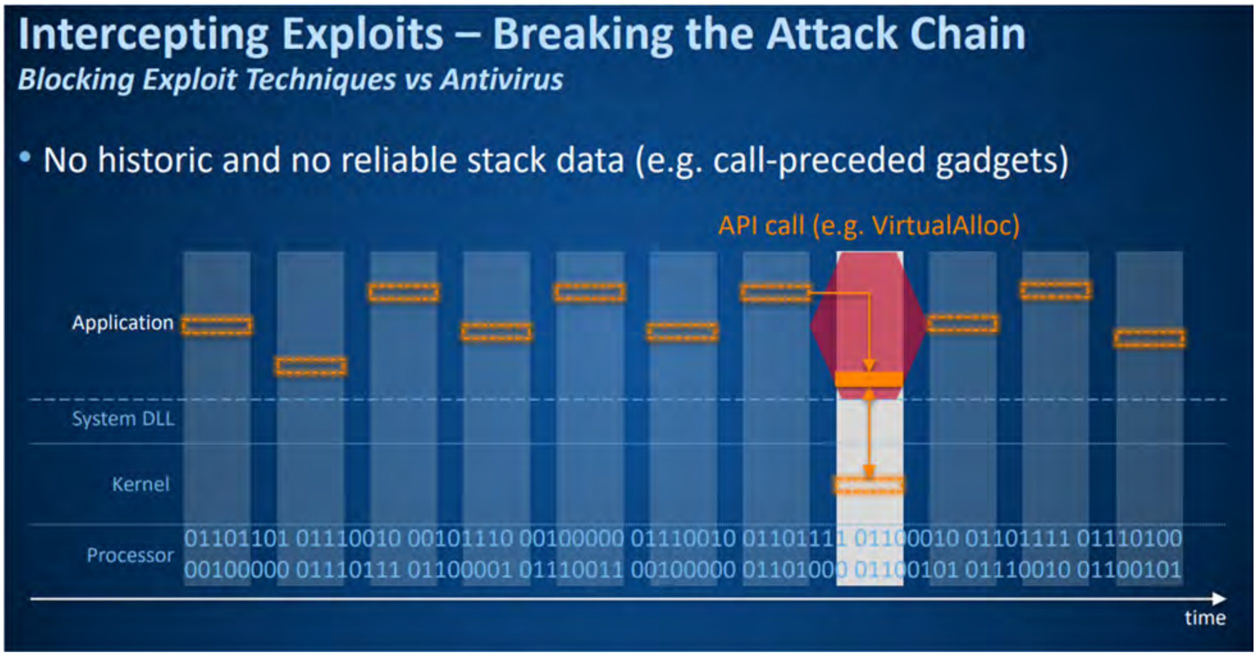
To defeat security technologies like data execution prevention (DEP) and address space layout randomization (ASLR), attackers typically resort to hijacking control-flow of vulnerable internet-facing applications. Such in-memory attacks are invisible to antivirus, most "next-gen" products, and other cyber defenses as there are no malicious files involved. Instead, the attack is constructed at run time by combining short pieces of benign code that are part of existing applications like Internet Explorer and Adobe Flash Player – a so-called code-reuse or return-oriented programming (ROP) attack.

During normal control-flow, sensitive API functions – like VirtualAlloc and CreateProcess – are invoked by the CALL instruction. Upon invoking a sensitive API, typical ROP defenses stop code execution to determine the API invoking address, using the 'return' address which is located on top of the stack. If the instruction of the API invoking address is not a CALL, the process is terminated.

Since the contents of the stack are writable, an attacker can write specific values on the stack to mislead the analysis of the stack-based ROP defense. The stack-based ROP defense cannot determine if the contents of the stack are benign or manipulated by an attacker.

(*See* WBR_SOP000387, https://www.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Comprehensive-Exploit-Prevention-wpna.pdf.)





(*See* WBR_SOP000566, https://secure2.sophos.com/it-it/medialibrary/PDFs/other/end-of-ransomware/MarkLomanSophosInterceptX.ashx.)

476. Additionally, Intercept X detects events that are associated with known "Mitre ATT&CK" techniques. Such attack techniques include "Native API," whereby "[a]dversaries interact with native OS application programming interface[s] to execute behaviors." These "native APIs" are "leveraged by the OS during system boot." Intercept X generates such detections by checking process and driver behavior against "threat classification rules."

Detections

Detections show you activity that you might need to investigate.

To see detections, go to **Threat Analysis Center** > **Detections**.

Detections identify activity on your devices that's unusual or suspicious but hasn't been blocked. They're different from events where we detect and block activity that we already know to be malicious.

We generate detections based on data that devices upload to the Sophos Data Lake.

We check that data against threat classification rules. When there's a match, we show a detection.

This page tells you how to use detections to look for potential threats.

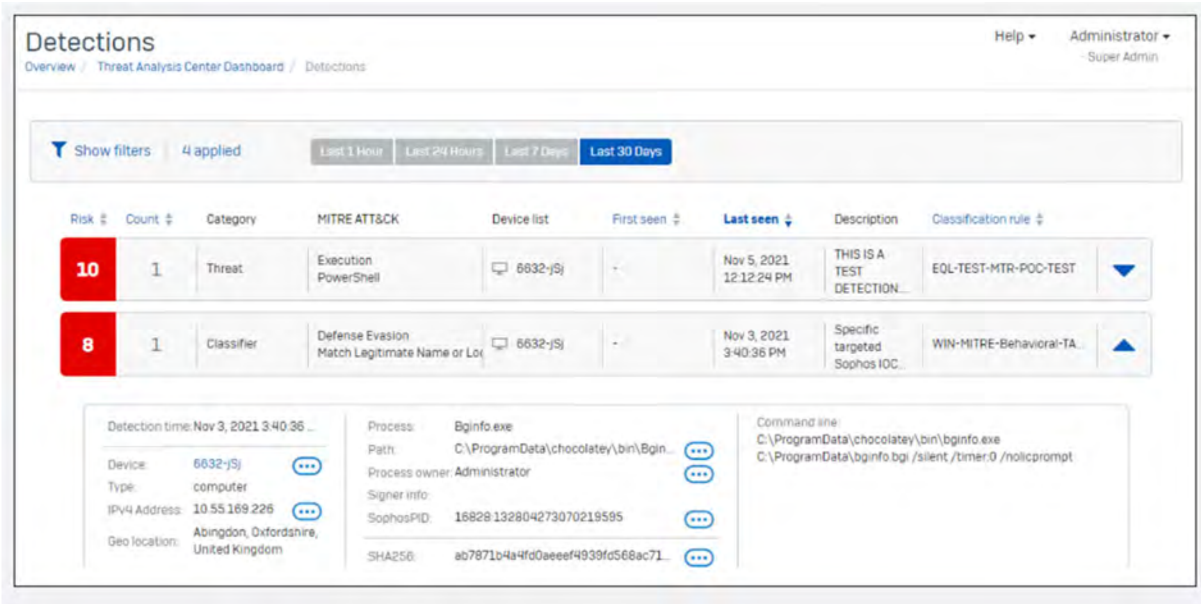
> **Note**
>
> **Investigations** can automatically group related detections together for more advanced analysis. See Investigations.

View detection details

To see detections, go to **Threat Analysis Center** > **Detections**.

We group detections according to the rule they matched and the date. The detection list shows the following:

- **Risk**. Risk is on a scale of 1 (lowest) to 10 (highest). With the default settings, we only show detections with a score of 7 or more. Use the score to prioritize investigations.
- **Classification rule**. The name of the rule that was matched.
- **Count**. Number of times the classification rule has been matched on a certain day.
- **Device list**. The device where the rule was last matched and the number of other devices with the same detection that day.
- **First seen** and **Last seen**. The first and last detections based on the classification rule that day.
- **Description**. What the rule identifies.
- **Mitre ATT&CK**. The corresponding Mitre ATT&CK Tactic and Technique.



(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/Detections/index.html.)

Native API

Adversaries may interact with the native OS application programming interface (API) to execute behaviors. Native APIs provide a controlled means of calling low-level OS services within the kernel, such as those involving hardware/devices, memory, and processes.[1][2] These native APIs are leveraged by the OS during system boot (when other system components are not yet initialized) as well as carrying out tasks and requests during routine operations.

(*See* https://attack.mitre.org/techniques/T1106/.)

477. The Accused Products also evaluate and trace a stack "[u]pon invoking a sensitive API…to determine the API invoking address, using the 'return' address which is located on top of the stack." In another example, the Accused Products include the "CallerCheck" anti-exploit module for "[p]revent[ing] API invocation from stack memory." (*See* WBR_SOP000444, https://news.sophos.com/en-us/2021/03/04/covert-code-faces-a-heap-of-trouble-in-memory/.)

478. Furthermore, Intercept X monitors API calls to identify and detect malicious activities. For example, attackers can attempt to load malicious libraries by placing them on UNC paths. By monitoring calls to the LoadLibrary API, Intercept X prevents this type of illicit library

loading.

> ## Load Library
>
> Attackers can attempt to load malicious libraries by placing them on UNC paths. Monitoring of all calls to the LoadLibrary API can be used to prevent this type of library loading.

(*See* https://assets.sophos.com/X24WTUEQ/at/cpbtrb4w4mfxqf4bn5cx8fh/sophos-comprehensive-exploit-prevention-wp.pdf.)

479. In addition, to preclude attackers from invoking the addresses of specific system functions through API calls (*e.g.* kernel32!VirtualProtect) via the "import address table" (IAT), Intercept X also filters out illicit calls by attackers to the IAT.

> ## Import Address Table Access Filtering (IAF)
>
> An attacker eventually needs the addresses of specific system functions (e.g. kernel32!VirtualProtect) to be able to perform malicious activities. These addresses can be retrieved from different sources, one of which is the import address table (IAT) of a loaded module. The IAT is used as a lookup table when an application calls a function in a different module. Because a compiled program cannot know the memory location of the libraries it depends upon, an indirect jump is required whenever an API call is made. As the dynamic linker loads modules and joins them together, it writes actual addresses into the IAT slots so that they point to the memory locations of the corresponding library functions.

(*See* https://assets.sophos.com/X24WTUEQ/at/cpbtrb4w4mfxqf4bn5cx8fh/sophos-comprehensive-exploit-prevention-wp.pdf)

480. The Accused Products perform a method that includes *generating, in response to the monitoring, a record of events, the record of events including the pestware-related events and analyzing the record of events so as to identify the pestware-related events and modifying the set of behavior rules so as to prevent the pestware related events.*

481. For example, Intercept X detects events that are associated with "threat classification rules" and known "Mitre ATT&CK" techniques. Detections are generated "based on data that devices upload to the Sophos Data Lake." Once data on process and driver behavior is uploaded, it is checked against Sophos' constantly updated threat classification rules. A subset of those rules, for example, corresponds to the "Native API" technique whereby "[a]dversaries interact with native OS application programming interface[s] to execute behaviors." Such "native APIs" are "leveraged by the OS during system boot." Intercept X generates such detections by checking process and driver behavior against "threat classification rules."

Detections

Detections show you activity that you might need to investigate.

To see detections, go to **Threat Analysis Center** > **Detections**.

Detections identify activity on your devices that's unusual or suspicious but hasn't been blocked. They're different from events where we detect and block activity that we already know to be malicious.

We generate detections based on data that devices upload to the Sophos Data Lake.

We check that data against threat classification rules. When there's a match, we show a detection.

This page tells you how to use detections to look for potential threats.

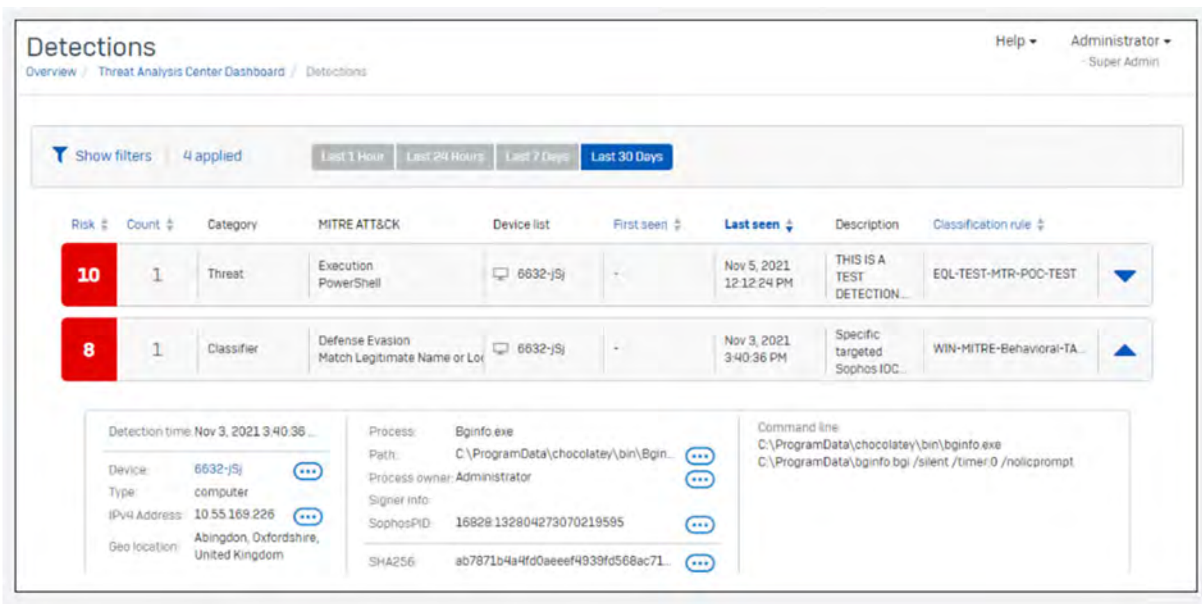
> **Note**
>
> **Investigations** can automatically group related detections together for more advanced analysis. See Investigations.

View detection details

To see detections, go to **Threat Analysis Center** > **Detections**.

We group detections according to the rule they matched and the date. The detection list shows the following:

- **Risk**. Risk is on a scale of 1 (lowest) to 10 (highest). With the default settings, we only show detections with a score of 7 or more. Use the score to prioritize investigations.
- **Classification rule**. The name of the rule that was matched.
- **Count**. Number of times the classification rule has been matched on a certain day.
- **Device list**. The device where the rule was last matched and the number of other devices with the same detection that day.
- **First seen** and **Last seen**. The first and last detections based on the classification rule that day.
- **Description**. What the rule identifies.
- **Mitre ATT&CK**. The corresponding Mitre ATT&CK Tactic and Technique.



(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/Detections/index.html.)

Native API

Adversaries may interact with the native OS application programming interface (API) to execute behaviors. Native APIs provide a controlled means of calling low-level OS services within the kernel, such as those involving hardware/devices, memory, and processes.[1][2] These native APIs are leveraged by the OS during system boot (when other system components are not yet initialized) as well as carrying out tasks and requests during routine operations.

(*See* https://attack.mitre.org/techniques/T1106/.)

482. The Accused Products perform a method that includes *scanning data in a registry of the computer for pestware during the second period in the boot sequence*. For example, Intercept X scans and protects "sensitive areas of the registry often used by adversaries to manipulate application and system behavior." Moreover, when a "detection of malicious activity occurs," Intercept X's "Sophos Clean" module "will be told to wipe the file and any of its associated registry entries, links, and files."

> **Registry Protections** – Protects sensitive areas of the registry often used by adversaries to manipulate application and system behavior.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

> A new quarantine has been created to hold convicted malware. When a detection of malicious activity occurs, Sophos Clean will be told to wipe the file and any of its associated registry entries, links, and files. The information is placed in quarantine and can be released by the administrator directly from the detection event in Sophos Central.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

483. Additionally, Intercept X's "Credential Theft Protection" features "look for unauthorized interactions" by credential theft tools with "LSASS (Local Security Authority Subsystem Service] memory" and the "Security Account Manager" database registry, or "SAM DB registry."

Credential Theft Prevention

Intercept X detects when an adversary-controlled process is attempting to extract user and administrator authentication credentials from a device. An adversary credential theft can target multiple operating system components to steal the password or the hashed passwords of users and administrators for the device. Dozens of different tools are available for the adversary to achieve this, but the most commonly used include Mimikatz, a credential extraction tool that targets LSASS (Local Security Authority Subsystem Service) memory, and hashdump, a credential theft tool that extracts the hashed password from the SAM (Security Account Manager) database.

How does Intercept X prevent credential theft?

Instead of targeting the specific tools used by adversaries (and there are a lot of them) Intercept X instead looks for unauthorized interactions with the LSASS runtime memory, the SAM DB registry, and direct extraction of credential data from the hard disk. As a prevention technique, we have tested with a variety of malware and penetration and hacking tools and found the mitigation to be extremely effective without generating false positive alerts for legitimate software that interacts with the LSASS and SAM DB.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

484. Each claim in the ’505 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ‘505 Patent.

485. Defendant directly infringes at least claim 1 of the ’505 Patent, either literally or under the doctrine of equivalents, by performing the steps described above. For example, Defendant performs the claimed method as described above by running the Sophos security software and corresponding systems to protect its own computer and network operations. Defendant also performs the claimed method as described above when testing the operation of the Accused Products and corresponding systems. As another example, Defendant performs the claimed method as described above when providing or administering services to third parties, customers, and partners using the Accused Products.

486. Defendant’s partners, customers, and end users of the Accused Products and corresponding systems and services directly infringe at least claim 1 of the ’505 Patent, literally or under the doctrine of equivalents, at least by using the Accused Products and corresponding

systems and services, as described above.

487. Defendant has actively induced and is actively inducing infringement of at least claim 1 of the ’505 Patent with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b). For example, Sophos encourages and induces customers with specific intent to use Sophos’s security software in a manner that infringes claim 1 of the ’505 Patent at least by offering and providing software that performs a method that infringes claim 1 when installed and operated by the customer, and by engaging in activities relating to selling, marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including the activities described below.

488. Defendant encourages, instructs, directs, and/or requires third parties—including its certified partners and/or customers to perform the claimed method using the software, services, and systems in infringing ways, as described above.

489. Defendant further encourages and induces its customers to infringe claim 1 of the ’505 Patent: 1) by making its security services available on its website, providing applications that allow users to access those services, widely advertising those services, and providing technical support and instructions to users, and 2) through activities relating to marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including its Sophos security software, and services in the United States. (*See* https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx; WBR_SOP000525, https://partners.sophos.com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1; WBR_SOP000209, https://secure2.sophos.com/en-us/security-news-trends/whitepapers/gated-wp/cybersecurity-system-buyers-guide.aspx%23formFrame.)

490. For example, Defendant shares instructions, guides, and manuals, which advertise

and instruct third parties on how to use the software as described above, including at least customers and partners. (*Id*.) Defendant provides customer service and technical support to purchasers of the Accused Products and corresponding system and services, which directs and encourages customers to perform certain actions that use the Accused Products in an infringing manner. (*Id*.)

491. Defendant and/or its partners recommend and sell the Accused Products and provide technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. Each customer enters into a contractual relationship with Defendant and/or one of its partners, which obligates each customer to perform certain actions in order to use the Accused Products. Further, in order to receive the benefit of Defendant's or its partner's continued technical support and their specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that performs the claimed method and infringes the '505 Patent.

492. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Product remains operational for each customer through ongoing technical support. Defendant and/or Sophos's partners affirmatively aid and abet each customer's use of the Accused Products in a manner that performs the claimed method of, and infringes, the '505 Patent.

493. Defendant also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the claimed methods, and not a staple article or commodity of commerce suitable for substantial non-infringing

uses. Indeed, as shown above, the Accused Products and the example functionality described below have no substantial non-infringing uses but are specifically designed to practice the ’505 Patent.

494. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the boot sequence malware detection and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. These processes are continually running when the system is in use and cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’505 Patent, that functionality could not be performed.

495. Additionally, the accused functionality, including boot sequence malware detection related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without extracting and analyzing a set of static features from an object and calibrating a feature vector on the basis of that analysis, the Accused Products could not deploy their machine learning model to determine whether or not that object is malicious or benign. These processes are continually running when the system is in use and cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the '505 Patent, that functionality could not be performed.

496. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Defendant constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionality constitute a material part of the inventions claimed at least because such an attack chain is integral to the processes identified above (such as "monitoring events…"; "managing pestware-related events…in a boot sequence of the computer…"; "managing pestware-related events in accordance with a set of behavior rules…"; "generating, in response to the monitoring, a record of events…"; "analyzing the record of events…"; "modifying the set of behavior rules…"; and "scanning data in a registry of the computer…") as recited in the claims of the '505 Patent. None of these products are staple goods—they are sophisticated and customized cyber security and malware detection products, methods, and systems.

497. Defendant's infringing actions have continued after this amended complaint was filed. Defendant had knowledge of the '505 Patent and of the specific acts that constitutes infringement of the '505 Patent at least on the date it received a copy of the draft complaint, but has continued to engage in the infringing activity, including by selling, making, configuring, and installing the Accused Products and performing the accused functionality, and by engaging in activities that constituted infringement and contributory infringement as described above.

498. The infringing actions of each partner, customer, and/or end-user of the Accused Products are attributable to Defendant. For example, Defendant directs and controls the activities or actions of its partners or others in connection with the Accused Products by contractual agreement or otherwise requiring partners or others to provide information and instructions to customers who acquire the Accused Products which, when followed, results in infringement.

Defendant further directs and controls the operation of devices executing the Accused Products by programming the software which, when executed by a customer or end user, perform the method steps of at least claim 1 of the ’505 Patent.

499. Plaintiffs have suffered and continue to suffer damages, including lost profits, as a result of Defendant’s infringement of the ’505 Patent. Defendant is therefore liable to Plaintiffs under 35 U.S.C. § 284 for damages in an amount that adequately compensates Plaintiffs for Defendant’s infringement, but no less than a reasonable royalty.

500. Plaintiffs will continue to suffer irreparable harm unless this Court preliminarily and permanently enjoins Defendant, its agents, employees, representatives, and all others acting in concert with Defendant from infringing the ’505 Patent.

501. Defendant’s infringement of the ’505 Patent is knowing and willful. Defendant had actual knowledge of the ’505 Patent at least when it received a copy of this amended complaint, and had constructive knowledge of the ’505 Patent from at least the date Plaintiffs marked their products with the ’505 Patent and/or provided notice of the ’505 Patent on their website.

502. Despite Defendant’s knowledge of the Asserted Patents and Plaintiffs’ patented technology, Defendant made the deliberate decision to sell products and services that they knew infringe the ’505 Patent. Defendant’s continued infringement of the ’505 Patent with knowledge of the ’505 Patent constitutes willful infringement.

TWELFTH CAUSE OF ACTION
(INFRINGEMENT OF THE ’932 PATENT)

503. Plaintiffs reallege and incorporate by reference the allegations of the preceding paragraphs of this Complaint.

504. Defendant has infringed and continues to infringe one or more claims of the ’932 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will

continue to do so unless enjoined by this Court. The Accused Products, including Intercept X, at least when used for their ordinary and customary purposes, practice each element of at least claim 1 of the ’932 Patent, as described below.

505. For example, claim 1 of the ’932 Patent recites:

> 1. A method for identifying an origin of activity on a computer that is indicative of pestware comprising:
>
> monitoring, using a kernel-mode driver, API call activity on the computer;
>
> storing information related to the API call activity in a log;
>
> analyzing, heuristically, the API call activity to determine whether one or more weighted factors associated with the API call activity exceeds a threshold;
>
> identifying, based upon the API call activity, a suspected pestware object on the computer;
>
> identifying, in response to the identifying the suspected pestware object, a reference to an identity of an externally networked source of the suspected pestware object; and
>
> reporting the identity of the externally networked source to an externally networked pestware research entity, wherein
>
> the identity of the externally networked source is selected from a group consisting of an I.P. address, a URL, an email client and a program.

506. The Accused Products perform each element of the method of claim 1 of the ’932 Patent. To the extent the preamble is construed to be limiting, the Accused Products *perform a method for identifying an origin of activity on a computer that is indicative of pestware*, as further explained below.

507. For example, Intercept X generates a root cause analysis to determine how the attacking process started and what else may have been happening on the device that is related to the root cause or detected escalation. The root cause analysis report includes detailed information of malicious activity and identifies how the attack unfolded and where it came from.

What happens when an attack is detected?

We will terminate the process and notify the end user. This will also initiate Sophos Clean to remove the malware. Upon detection, a root cause analysis will be generated to determine how the attacking process started and what else may have been happening on the device that is related to the root cause or detected escalation. The endpoint will be put into a red security health state, as this attack indicates an adversary has likely penetrated the device and more investigation is recommended.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

What should an admin do?

Like similar exploit prevention detections, administrators should review the root cause analysis report to determine how the attack unfolded and where it came from. Once the investigation is complete the administrator can clear the alert to allow normal operation of the device.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

Endpoint Detection and Response (EDR)

When malicious activity is detected on a device it is critical that the administrator has more information so that they can understand the nature of the incident, how it happened, what lead to the detection, and what actions that should be taken to prevent similar incidents in the future. This ability to perform forensics on an attack has been the work of security operations centers armed with SIEM and device forensic tools. Unfortunately, the volume of malicious activity detections in a typical organization can easily overwhelm understaffed IT security administrators. To address this challenge an expanded version of Intercept X, Intercept X Advanced with EDR, is available.

Top RCA questions

Is EDR available as a stand-alone product?

No, EDR capabilities are only included in Intercept X Advanced with EDR. This product combines all the capabilities of Intercept X and Central Endpoint Protection with EDR functionality.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

Malware expertise: Most organization rely on malware experts that specialize in reverse engineering to analyze suspicious files. Not only is this approach time consuming and difficult to achieve, but it assumes a level of cybersecurity sophistication which most organizations don't possess. Intercept X Advanced with EDR offers a better approach by leveraging Deep Learning Malware Analysis which automatically analyzes malware in extreme detail, breaking down file attributes and code and comparing them to millions of other files. Analysts can easily see which attributes and code segments are similar to "known-good" and "known bad" files so they can determine if a file should be blocked or allowed.

(*See* https://www.firewalls.com/pub/media/wysiwyg/datasheets/Sophos/intercept-x-edr.pdf.)

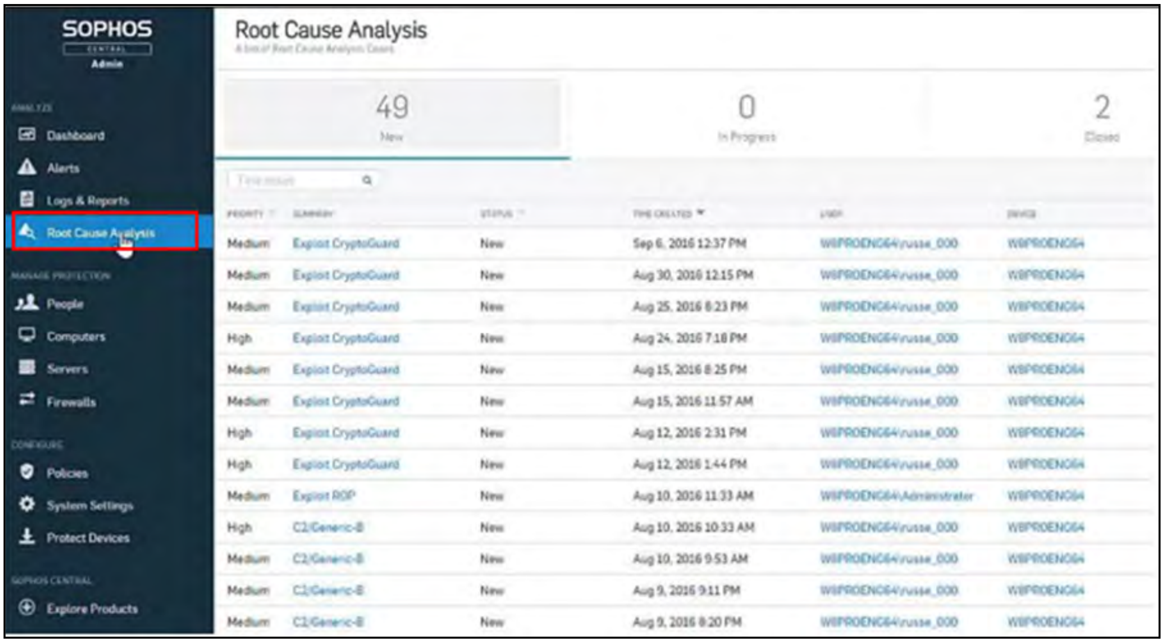
	Sophos Intercept X Advanced with EDR	Sophos Intercept X Advanced	Sophos Intercept X	Sophos Endpoint Protection
Foundational techniques	✓	✓		✓
Deep learning	✓	✓	✓	
Anti-exploit	✓	✓	✓	
CryptoGuard anti-ransomware	✓	✓	✓	
Endpoint detection and response (EDR)	✓			

(*See* https://www.firewalls.com/pub/media/wysiwyg/datasheets/Sophos/intercept-x-edr.pdf.)

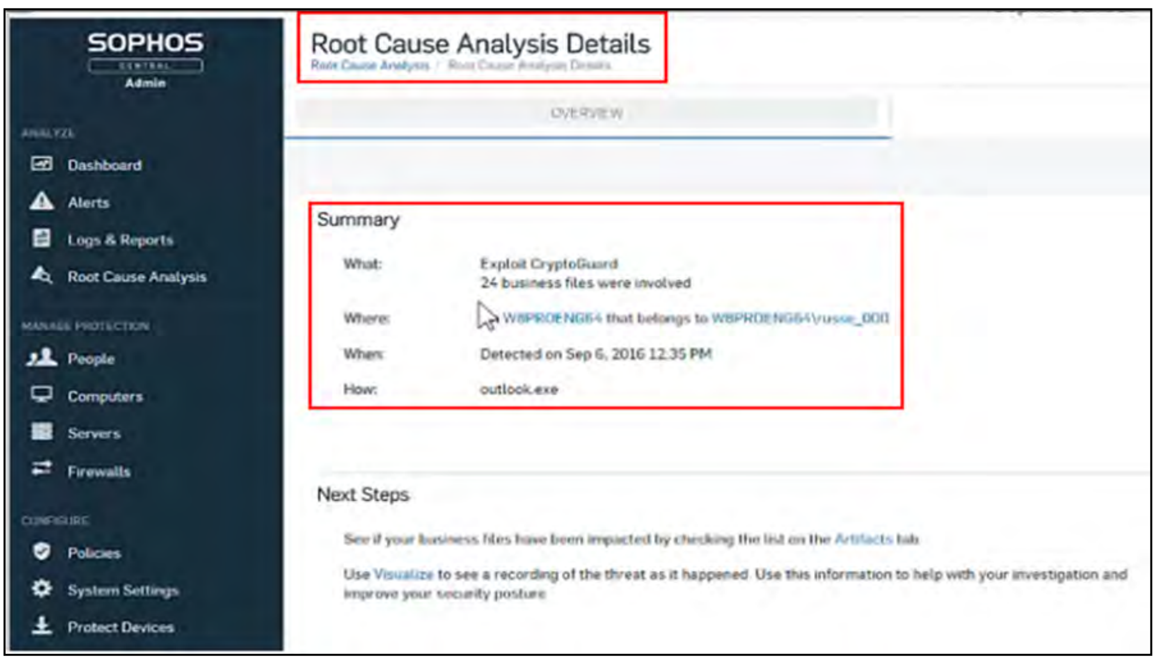
Sophos Intercept X
Sophos Intercept X Advanced with EDR
Sophos Intercept X for Server

Get the root cause analysis of an attack with complete visibility on the how and where of the attack along with recommendations on what your next steps should be.

(*See* https://assets.sophos.com/X24WTUEQ/at/pcxhcspjp7w9f79m4xqg4zq3/sophos-nist-compliance-card.pdf.)



(*See* https://www.sophos.com/en-us/medialibrary/Files/ppt/Solution%20partner%20event/ SolPartner_Intercept X.pdf?la=en.pptx.)



(*See* https://www.sophos.com/en-us/medialibrary/Files/ppt/Solution%20partner%20event/ SolPartner_Intercept X.pdf?la=en.pptx.)

Sophos Central Intercept X Advanced combines endpoint protection/detection and response into a single solution. It combines AI, behavioral analysis and tight exploit protection into a deep defense that can thwart most malware attacks. Let's break down Intercept X Advanced with EDR and see what sets it apart from the competition.

(*See* https://www.corporatearmor.com/product/sophos-central-intercept-x-advanced-with-edr-for-25-49-users-3-year-must-purchase-a-minimum-qty-of-25/.)

508. In addition, one of the core components of the root cause analysis is "threat case" assessment. Detection of malicious activity, including malware, exploits, ransomware detections, and blocking activity will trigger Intercept X's threat case algorithm to examine the contents of the data recorder. The objective of the algorithm is to take an identified malicious action, called a "beacon event," and then track events associated with the beacon back to an origin or root cause. The root cause can, for example, be the opening of an email attachment, the insertion of a USB drive, browsing to a website, and other similar activity. Once the root cause has been detected, the algorithm then moves forward from that event and identifies associated activity.

Threat cases – Detection of malicious activity, including malware, exploits, ransomware detections, and blocking activity will trigger the threat case algorithm to examine the contents of the data recorder. The objective of the algorithm is to take an identified malicious action, called a beacon event, and then track events associated with the beacon back to an origin or root cause. The root cause can be things like opening an email attachment, inserting a USB drive, browsing to website, and other activity. Once the root cause has been detected, the algorithm then moves forward from that event and identifies associated activity. Analysts can choose to view the full threat case, or the direct path only. By automatically tracking to the root cause and collecting associated data, Sophos is able to perform much of the work that a manual forensic examination would need to do. This dramatically reduces the time and expertise required to understand the origin of malicious activity.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

509. The Accused Products perform a method that includes *monitoring, using a kernel-mode driver, API call activity on the computer* and *storing information related to the API call activity in a log*. Intercept X monitors classes of applications at the kernel level. This injection into the process tree allows close and continuous monitoring of activities engaged in by a given process, including memory access, disk access, network access, DLLs loaded, and other process interactions.

510. Furthermore, Intercept X uses a "data recorder" that tracks activity on each device. The data recorder collects information on processes, memory, network, disk, and registry changes and both records that information locally on the device and makes it available over the cloud for "threat case" generation when an incident is detected. Activity over the last 30 days on the device is stored by the data recorder.

How does Intercept X prevent vulnerabilities being?

Intercept X monitors classes of applications at the kernel level. This injection into the process allows close and continuous monitoring of activity in the process, including memory access, disk, network access, DLLs loaded, and other process interactions.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

Sophos Intercept X includes a novel approach to prevent attackers from addressing sensitive kernel functions via API functions that are unprotected.

(*See* https://assets.sophos.com/X24WTUEQ/at/cpbtrb4w4mfxqf4bn5cx8fh/sophos-comprehensive-exploit-prevention-wp.pdf.)

Root cause analysis core components

Sophos data recorder – Intercept X includes a data recorder that tracks activity on the device. Information collected on process, memory, network, disk, and registry changes are recorded locally on the device and made available for threat case generation when an incident is detected. Activity over the last 30 days on the device is stored and consumes about 100MB of disk space.

Threat cases – Detection of malicious activity, including malware, exploits, ransomware detections, and blocking activity will trigger the threat case algorithm to examine the contents of the data recorder. The objective of the algorithm is to take an identified malicious action, called a beacon event, and then track events associated with the beacon back to an origin or root cause. The root cause can be things like opening an email attachment, inserting a USB drive, browsing to website, and other activity. Once the root cause has been detected, the algorithm then moves forward from that event and identifies associated activity. Analysts can choose to view the full threat case, or the direct path only. By automatically tracking to the root cause and collecting associated data, Sophos is able to perform much of the work that a manual forensic examination would need to do. This dramatically reduces the time and expertise required to understand the origin of malicious activity.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

511. Moreover, the Accused Products monitor "sensitive API functions," include "Memory Scanning" for "defense against in-memory malware" and monitor "API call[s] (*e.g.*, VitrualAlloc)."

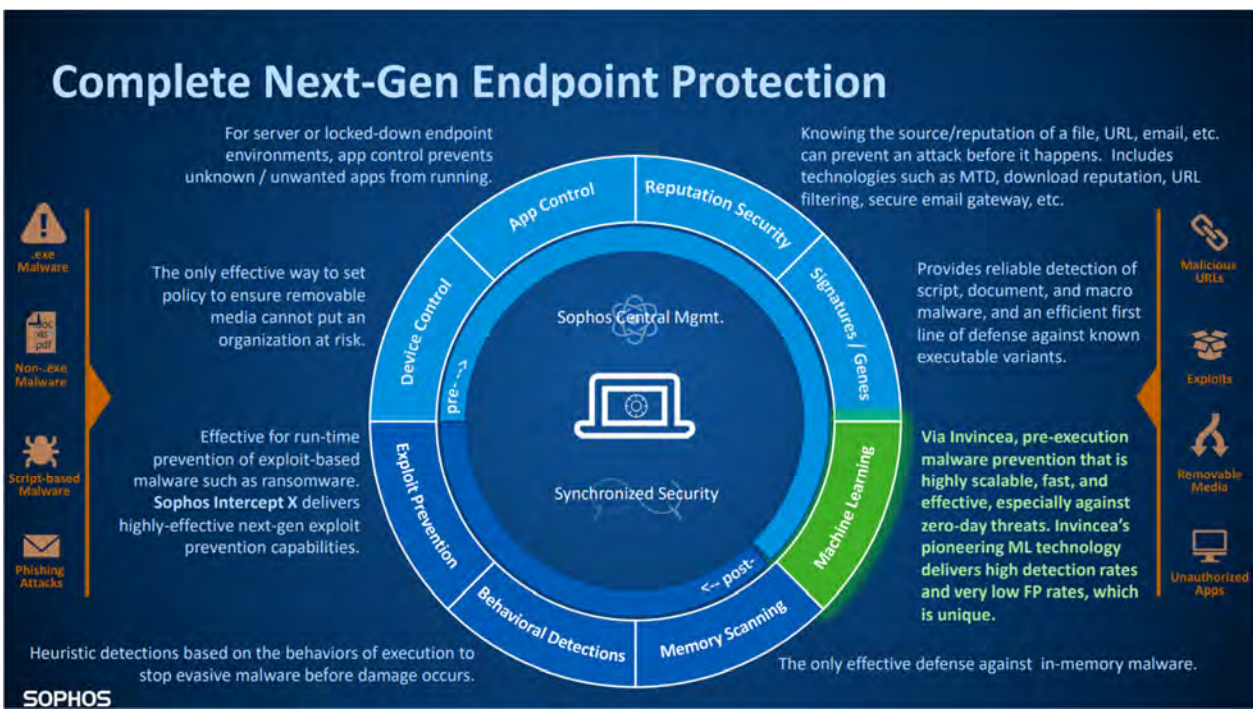
Stack-based ROP Mitigation (Caller)

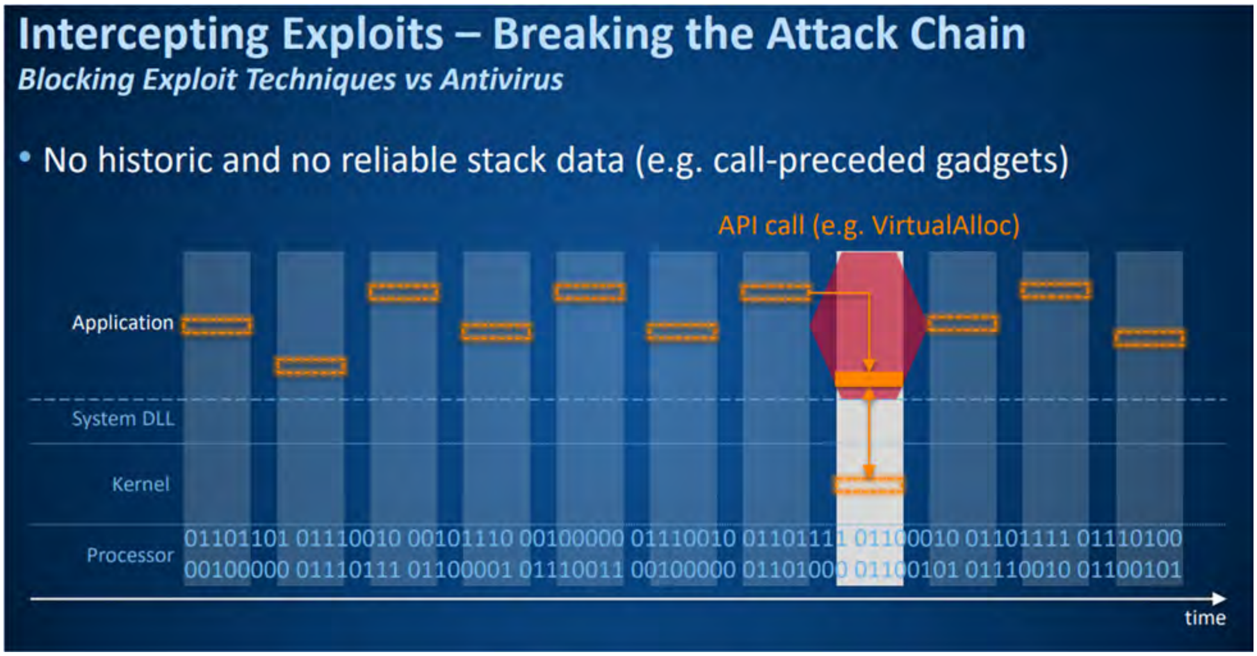
To defeat security technologies like data execution prevention (DEP) and address space layout randomization (ASLR), attackers typically resort to hijacking control-flow of vulnerable internet-facing applications. Such in-memory attacks are invisible to antivirus, most "next-gen" products, and other cyber defenses as there are no malicious files involved. Instead, the attack is constructed at run time by combining short pieces of benign code that are part of existing applications like Internet Explorer and Adobe Flash Player – a so-called code-reuse or return-oriented programming (ROP) attack.

During normal control-flow, sensitive API functions – like VirtualAlloc and CreateProcess – are invoked by the CALL instruction. Upon invoking a sensitive API, typical ROP defenses stop code execution to determine the API invoking address, using the 'return' address which is located on top of the stack. If the instruction of the API invoking address is not a CALL, the process is terminated.

Since the contents of the stack are writable, an attacker can write specific values on the stack to mislead the analysis of the stack-based ROP defense. The stack-based ROP defense cannot determine if the contents of the stack are benign or manipulated by an attacker.

(*See* WBR_SOP000387, https://www.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Comprehensive-Exploit-Prevention-wpna.pdf.)





(*See* WBR_SOP000566, https://secure2.sophos.com/it-it/medialibrary/PDFs/other/end-of-ransomware/MarkLomanSophosInterceptX.ashx.)

512. The Accused Products also evaluate and trace a stack "[u]pon invoking a sensitive API…to determine the API invoking address, using the 'return' address which is located on top of the stack." In another example, the Accused Products include the "CallerCheck" anti-exploit module for "[p]revent[ing] API invocation from stack memory." (*See* WBR_SOP000444, https://news.sophos.com/en-us/2021/03/04/covert-code-faces-a-heap-of-trouble-in-memory/.)

513. Furthermore, Intercept X monitors API calls to identify and detect malicious activities. For example, attackers can attempt to load malicious libraries by placing them on UNC paths. By monitoring calls to the LoadLibrary API, Intercept X prevents this type of illicit library loading.

Load Library

Attackers can attempt to load malicious libraries by placing them on UNC paths. Monitoring of all calls to the LoadLibrary API can be used to prevent this type of library loading.

(*See* https://assets.sophos.com/X24WTUEQ/at/cpbtrb4w4mfxqf4bn5cx8fh/sophos-comprehensive-exploit-prevention-wp.pdf.)

514. In addition, to preclude attackers from invoking the addresses of specific system functions through API calls (*e.g*. kernel32!VirtualProtect) via the "import address table" (IAT), Intercept X also filters out illicit calls by attackers to the IAT.

Import Address Table Access Filtering (IAF)

An attacker eventually needs the addresses of specific system functions (e.g. kernel32!VirtualProtect) to be able to perform malicious activities. These addresses can be retrieved from different sources, one of which is the import address table (IAT) of a loaded module. The IAT is used as a lookup table when an application calls a function in a different module. Because a compiled program cannot know the memory location of the libraries it depends upon, an indirect jump is required whenever an API call is made. As the dynamic linker loads modules and joins them together, it writes actual addresses into the IAT slots so that they point to the memory locations of the corresponding library functions.

(*See* https://assets.sophos.com/X24WTUEQ/at/cpbtrb4w4mfxqf4bn5cx8fh/sophos-comprehensive-exploit-prevention-wp.pdf.)

515. The Accused Products perform a method that includes *analyzing, heuristically, the API call activity to determine whether one or more weighted factors associated with the API call activity exceeds a threshold and identifying, based upon the API call activity, a suspected pestware object on the computer*. For example, Intercept X's Threat Analysis Center assigns a "Risk" score to each activity it has detected as "unusual or suspicious" but has not yet blocked, and only displays

such detections by default when they have a score equal to or greater than seven. Whether or not the detected threat is associated with a known "Mitre ATT&CK" technique increases its risk score. Such attack techniques include "Native API," whereby "[a]dversaries interact with native OS application programming interface[s] to execute behaviors."

Detections

Detections show you activity that you might need to investigate.

To see detections, go to **Threat Analysis Center** > **Detections**.

Detections identify activity on your devices that's unusual or suspicious but hasn't been blocked. They're different from events where we detect and block activity that we already know to be malicious.

We generate detections based on data that devices upload to the Sophos Data Lake.

We check that data against threat classification rules. When there's a match, we show a detection.

This page tells you how to use detections to look for potential threats.

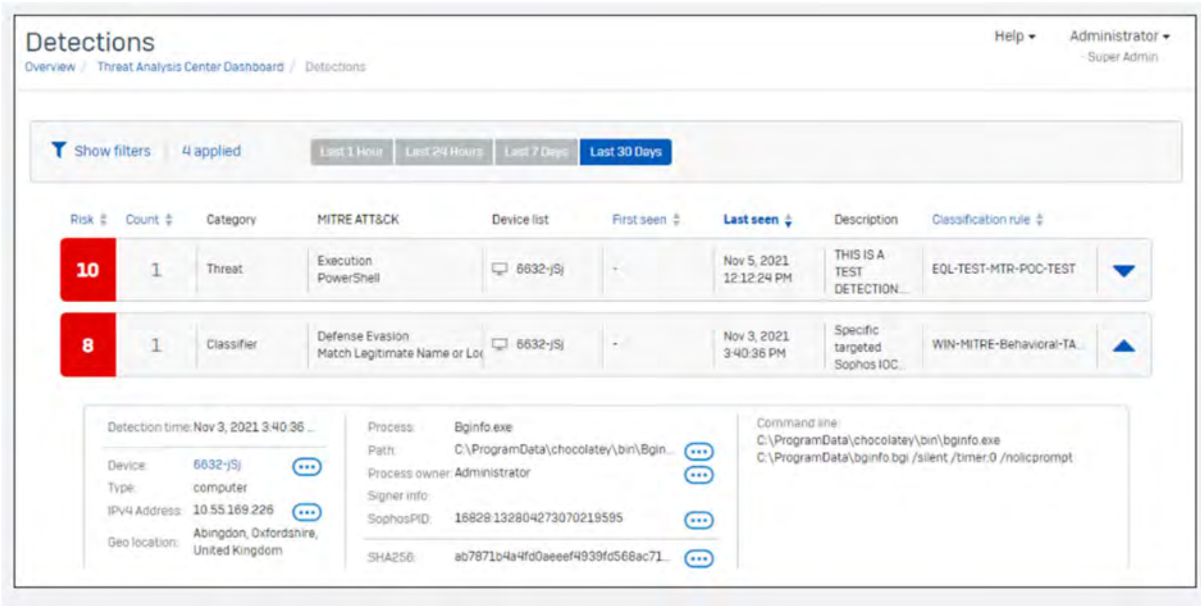
> **Note**
>
> **Investigations** can automatically group related detections together for more advanced analysis. See Investigations.

View detection details

To see detections, go to **Threat Analysis Center** > **Detections**.

We group detections according to the rule they matched and the date. The detection list shows the following:

- **Risk**. Risk is on a scale of 1 (lowest) to 10 (highest). With the default settings, we only show detections with a score of 7 or more. Use the score to prioritize investigations.
- **Classification rule.** The name of the rule that was matched.
- **Count**. Number of times the classification rule has been matched on a certain day.
- **Device list**. The device where the rule was last matched and the number of other devices with the same detection that day.
- **First seen** and **Last seen**. The first and last detections based on the classification rule that day.
- **Description**. What the rule identifies.
- **Mitre ATT&CK**. The corresponding Mitre ATT&CK Tactic and Technique.



(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/Detections/index.html.)

Native API

Adversaries may interact with the native OS application programming interface (API) to execute behaviors. Native APIs provide a controlled means of calling low-level OS services within the kernel, such as those involving hardware/devices, memory, and processes.[1][2] These native APIs are leveraged by the OS during system boot (when other system components are not yet initialized) as well as carrying out tasks and requests during routine operations.

(*See* https://attack.mitre.org/techniques/T1106/.)

516. Furthermore, the root cause analysis generated by Intercept X determines how the attacking process started and what else may have been happening on the device that is related to the root cause or detected escalation. The root cause analysis report includes detailed information of malicious activity and, for instance, identifies how the attack unfolded and where it came from.

What happens when an attack is detected?

We will terminate the process and notify the end user. This will also initiate Sophos Clean to remove the malware. Upon detection, a root cause analysis will be generated to determine how the attacking process started and what else may have been happening on the device that is related to the root cause or detected escalation. The endpoint will be put into a red security health state, as this attack indicates an adversary has likely penetrated the device and more investigation is recommended.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

What should an admin do?

Like similar exploit prevention detections, administrators should review the root cause analysis report to determine how the attack unfolded and where it came from. Once the investigation is complete the administrator can clear the alert to allow normal operation of the device.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

Endpoint Detection and Response (EDR)

When malicious activity is detected on a device it is critical that the administrator has more information so that they can understand the nature of the incident, how it happened, what lead to the detection, and what actions that should be taken to prevent similar incidents in the future. This ability to perform forensics on an attack has been the work of security operations centers armed with SIEM and device forensic tools. Unfortunately, the volume of malicious activity detections in a typical organization can easily overwhelm understaffed IT security administrators. To address this challenge an expanded version of Intercept X, Intercept X Advanced with EDR, is available.

Top RCA questions

Is EDR available as a stand-alone product?

No, EDR capabilities are only included in Intercept X Advanced with EDR. This product combines all the capabilities of Intercept X and Central Endpoint Protection with EDR functionality.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf.)

Malware expertise: Most organization rely on malware experts that specialize in reverse engineering to analyze suspicious files. Not only is this approach time consuming and difficult to achieve, but it assumes a level of cybersecurity sophistication which most organizations don't possess. Intercept X Advanced with EDR offers a better approach by leveraging Deep Learning Malware Analysis which automatically analyzes malware in extreme detail, breaking down file attributes and code and comparing them to millions of other files. Analysts can easily see which attributes and code segments are similar to "known-good" and "known bad" files so they can determine if a file should be blocked or allowed.

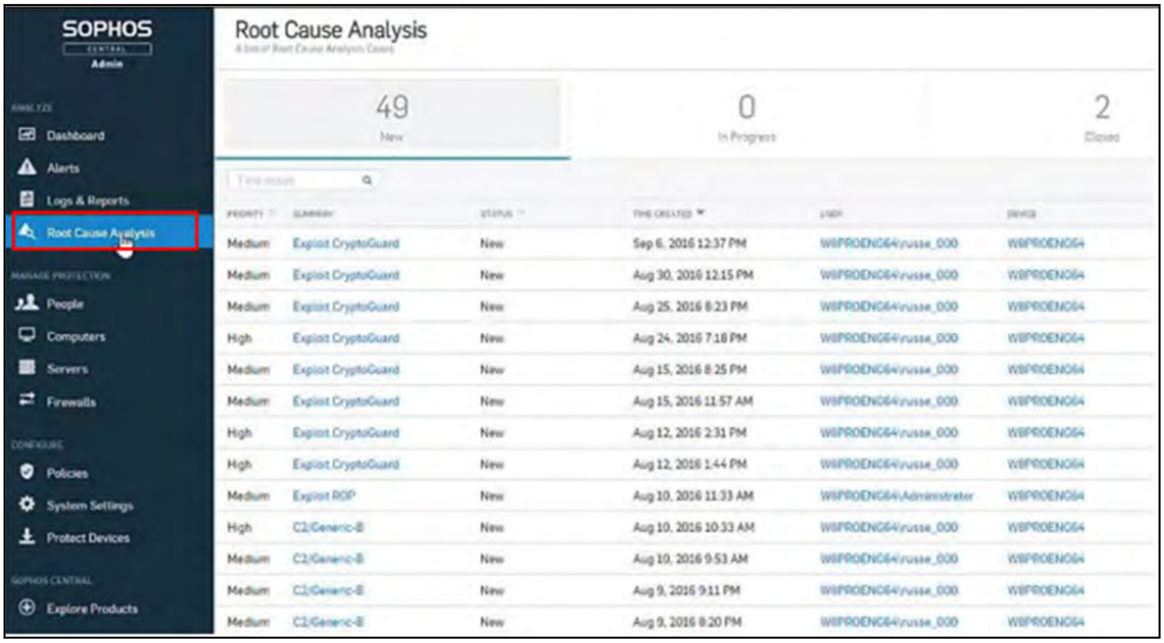
(*See* https://www.firewalls.com/pub/media/wysiwyg/datasheets/Sophos/intercept-x-edr.pdf.)

	Sophos Intercept X Advanced with EDR	Sophos Intercept X Advanced	Sophos Intercept X	Sophos Endpoint Protection
Foundational techniques	✓	✓		✓
Deep learning	✓	✓	✓	
Anti-exploit	✓	✓	✓	
CryptoGuard anti-ransomware	✓	✓	✓	
Endpoint detection and response (EDR)	✓			

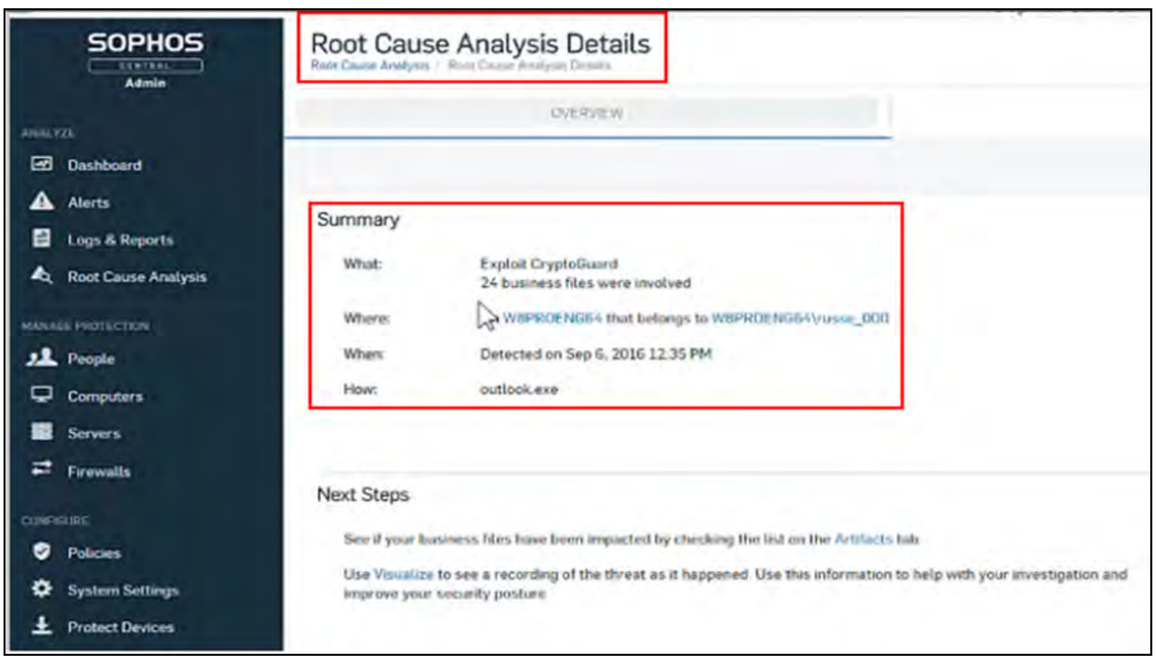
(*See* https://www.firewalls.com/pub/media/wysiwyg/datasheets/Sophos/intercept-x-edr.pdf.)

Sophos Intercept X Sophos Intercept X Advanced with EDR Sophos Intercept X for Server	Get the root cause analysis of an attack with complete visibility on the how and where of the attack along with recommendations on what your next steps should be.

(*See* https://assets.sophos.com/X24WTUEQ/at/pcxhcspjp7w9f79m4xqg4zq3/sophos-nist-compliance-card.pdf.)



(*See* https://www.sophos.com/en-us/medialibrary/Files/ppt/Solution%20partner%20event/ SolPartner_Intercept X.pdf?la=en.pptx.)



(*See* https://www.sophos.com/en-us/medialibrary/Files/ppt/Solution%20partner%20event/ SolPartner_Intercept X.pdf?la=en.pptx.)

Sophos Central Intercept X Advanced combines endpoint protection/detection and response into a single solution. It combines AI, behavioral analysis and tight exploit protection into a deep defense that can thwart most malware attacks. Let's break down Intercept X Advanced with EDR and see what sets it apart from the competition.

(*See* https://www.corporatearmor.com/product/sophos-central-intercept-x-advanced-with-edr-for-25-49-users-3-year-must-purchase-a-minimum-qty-of-25/.)

On-demand SophosLabs threat intelligence – When Intercept X Advanced with EDR elevates a potentially suspicious file, IT administrators can gather more information by accessing on-demand threat intelligence curated by SophosLabs which receives and processes approximately 400,000 previously unseen malware samples each day. This, and other threat intelligence is collected, aggregated, and summarized for easy analysis. This means that teams that do not have dedicated threat intelligence analysts, or access to expensive and hard to understand threat feeds, can benefit from one of the top cybersecurity research and data science teams in the world.

Automated malware analysis - Intercept X Advanced with EDR leverages Deep Learning Malware Analysis which automatically analyzes malware in extreme detail, breaking down file attributes, code similarity, and file path and comparing them to millions of other files. Analysts can easily see which attributes are similar to "known-good" and "known bad" files so they can determine if a file should be blocked or allowed.

(*See* https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Intercept-X-Solution-Brief.pdf)

517. The Accused Products perform a method that includes *identifying, in response to the identifying the suspected pestware object, a reference to an identity of an externally networked source of the suspected pestware object*.

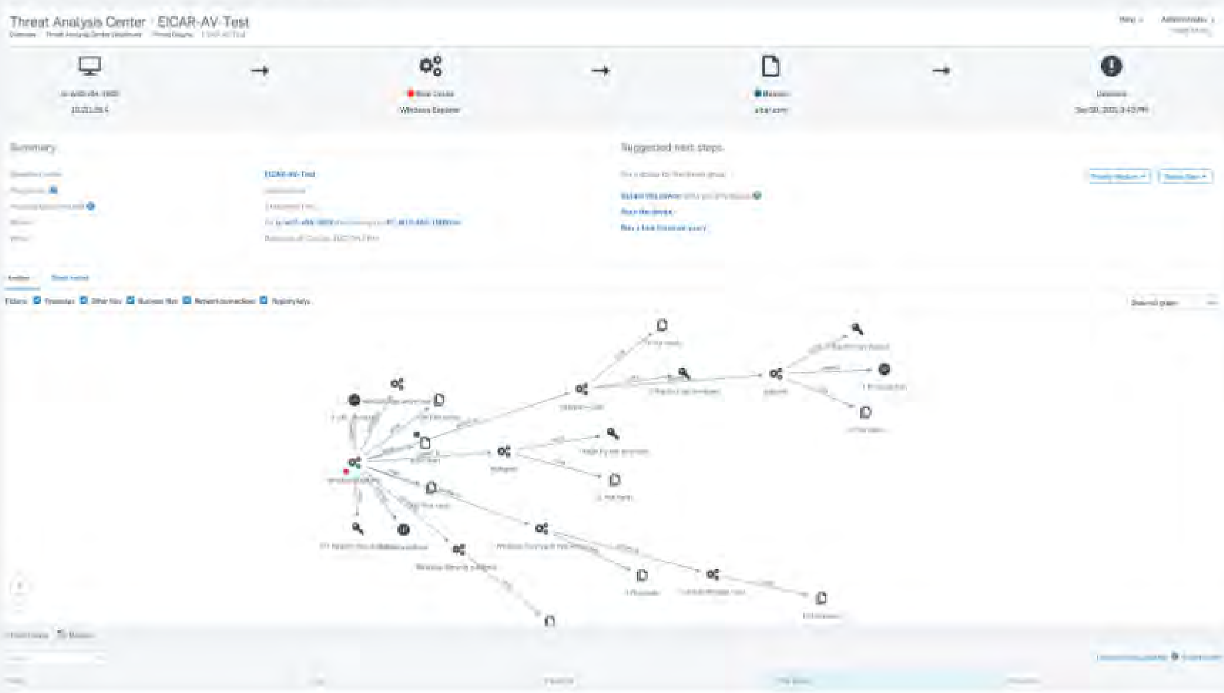
518. For example, Intercept X's Threat Analysis Center dashboard includes "Threat Graphs" for investigating and cleaning up malware attacks. Users can "investigate a threat graph by going to its details pages and using the analysis tools." For example, the threat summary provides a summary of threat activity, including the root cause, "Where" the threat was detected (*e.g.* on which device), and "When" the threat was detected. The "Threat Graph analysis" also identifies a reference to an identity of an externally networked source, such as "EICAR.com" in

the example below, that is suspected of originating pestware. The threat graph analysis also shows the chain of events in the malware infection. Moreover, the threat graph analysis' "Graph record tab shows the history of the threat graph, from its creation by Sophos or the admin."

Threat Graph analysis

You can investigate a threat graph by going to its details page and using the analysis tools there.

Find the threat graph on the **Threat Graphs** page. Click its name to see a simplified event chain, summary, details of the artifacts (processes, files, keys) affected, and a diagram showing how the threat developed. The following screenshot shows an Eicar-AV-Test detection triggered by opening a file `eicar.com` in Windows Explorer.



(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

Summary

The **Summary** shows you a summary of the threat, including these details:

- **Root cause**: Where the infection entered your system.
- **Possible data involved**: Files that might contain important data. Check them to see if data has been encrypted or stolen.
- **Where**: Name of the device and its user.
- **When**: Detection time and date.

Analyze

The **Analyze** tab shows the chain of events in the malware infection.

A menu on the right of the tab lets you choose how much detail you see:

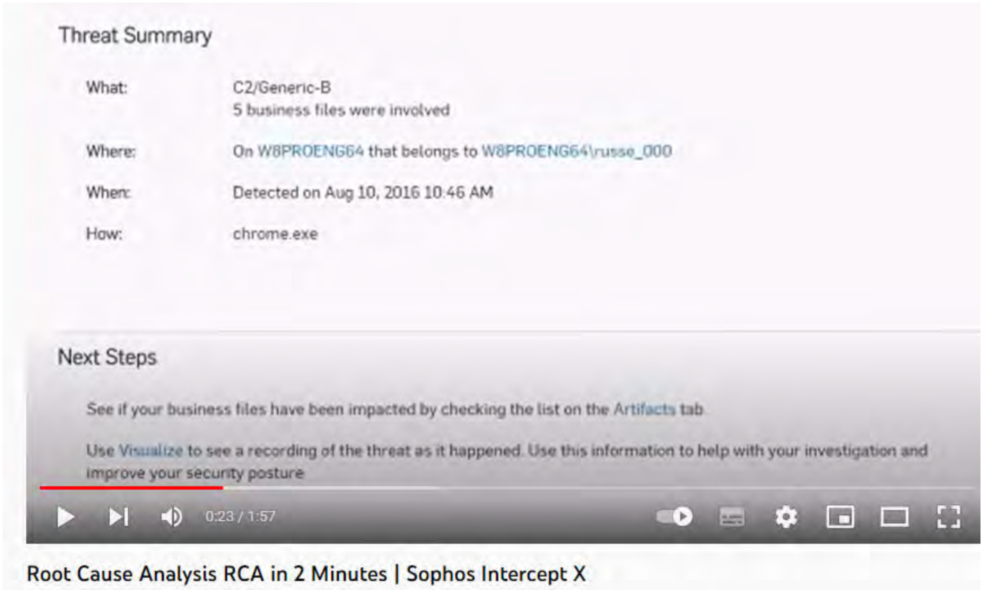
- **Show direct path**: This shows the chain of directly-involved items between the root cause and the item where the infection was detected (the "beacon").
- **Show full graph**: This shows the root cause, beacon, affected artifacts (applications, files, keys), the path of the infection (shown by arrows), and how the infection occurred. This is the default setting.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Overview/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

Graph record

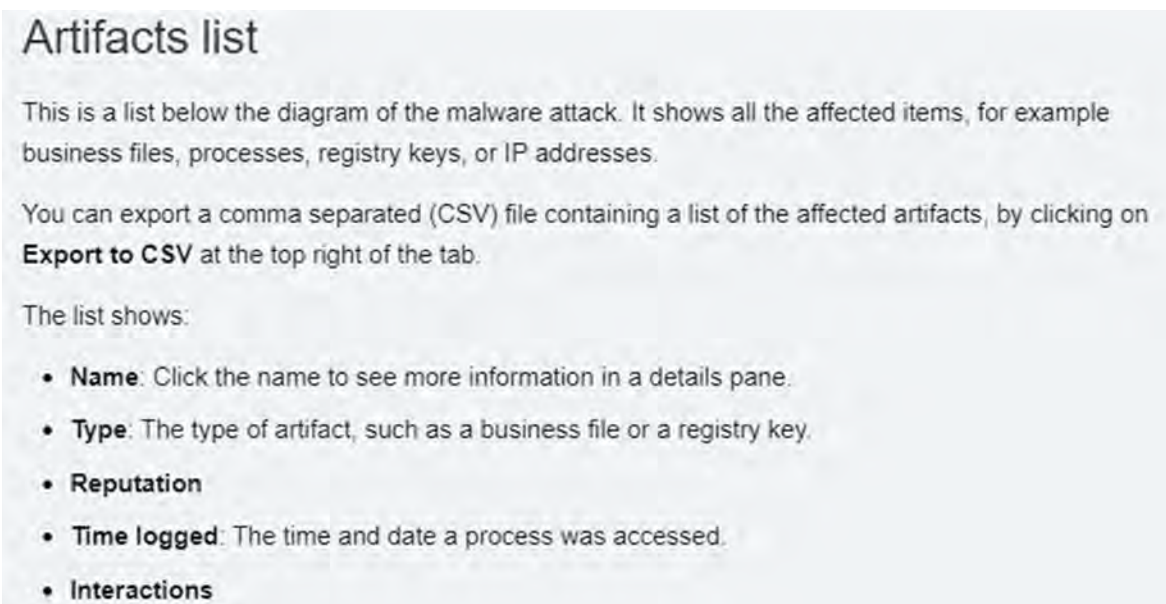
The **Graph record** tab shows the history of the threat graph, from its creation by Sophos or the admin. You can post comments to record actions that have been taken and other relevant information.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)



(*See* https://www.youtube.com/watch?v=AOsjUjp4P7Q.)

519. In another example, Intercept X's Threat Analysis Center dashboard includes artifacts list to show "all affected items, for example business files, processes, registry keys, or IP addresses."



Artifacts list

This is a list below the diagram of the malware attack. It shows all the affected items, for example business files, processes, registry keys, or IP addresses.

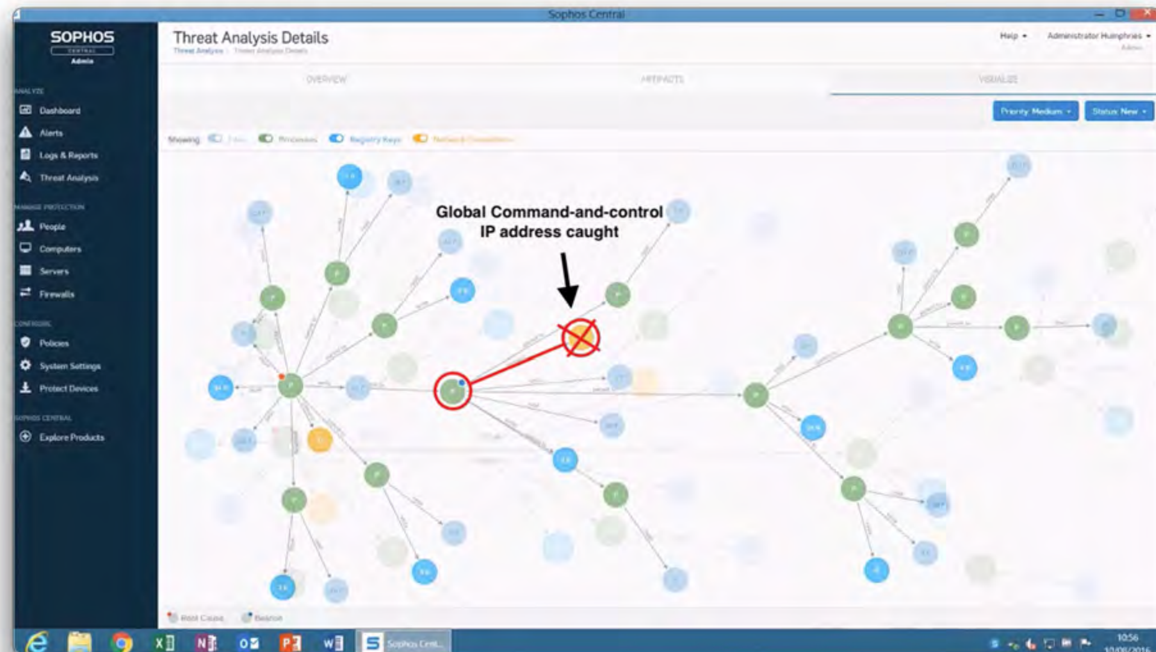
You can export a comma separated (CSV) file containing a list of the affected artifacts, by clicking on **Export to CSV** at the top right of the tab.

The list shows:

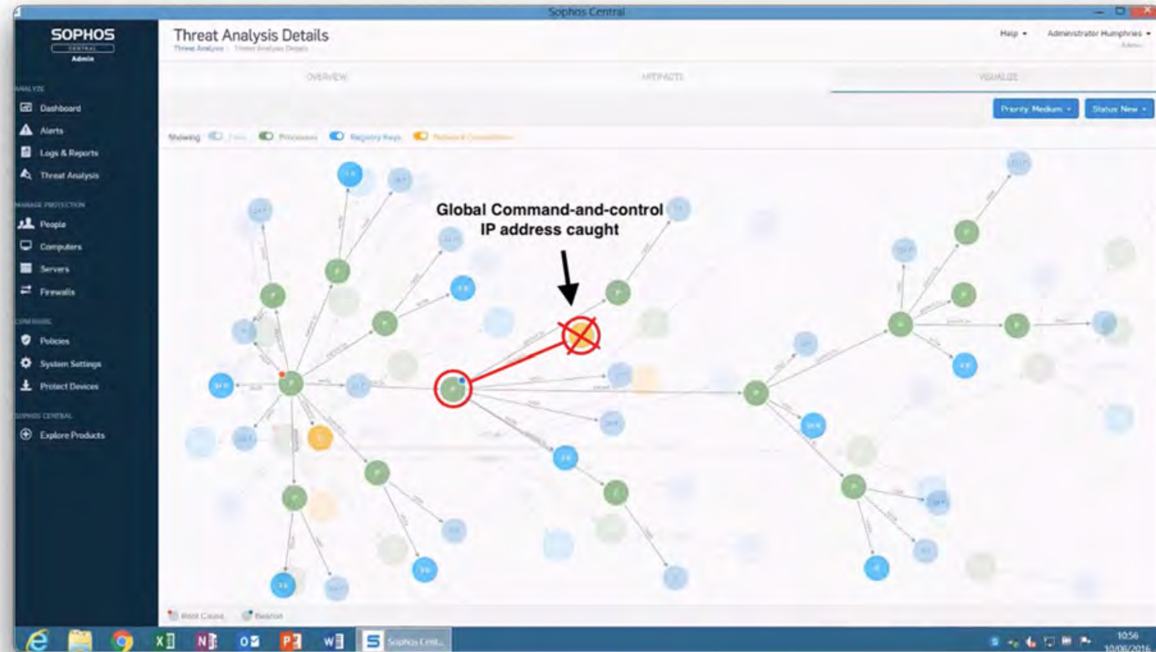
- **Name**: Click the name to see more information in a details pane.
- **Type**: The type of artifact, such as a business file or a registry key.
- **Reputation**
- **Time logged**: The time and date a process was accessed.
- **Interactions**

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

520. Moreover, For example, Intercept X's root cause analyses identify the IP addresses that originated each suspicious, or "beacon," event, and flag them as being "command-and-control" sites for malicious software if they are detected as being so by Sophos' threat analysis servers: "Here's the beacon event, the thing that was caught, in this case a fake salary report. Here's the root, Google Chrome. We see here that the fake report reached out to an IP address that we happen to know to be a command-and-control site. This right here is the moment of conviction, when we discovered that what was executing was a piece of malware."



(*See* https://youtu.be/AOsjUjp4P7Q?t=48.)



(*See* https://youtu.be/AOsjUjp4P7Q?t=48.)

521. Intercept X's Threat Analysis Center dashboard also generates a "forensic snapshot" from data it receives from the kernel-mode driver. The forensic snapshot is analyzed to identify the malicious activities.

Create forensic snapshot

You can create a "forensic snapshot" of data from the device. This gets data from a Sophos log of the device's activity and saves it on that device. For more information on forensic snapshots see Forensic snapshots.

You can also save it in the Amazon Web Services (AWS) S3 bucket you specify. You can then do your analysis.

You'll need a converter (which we provide) to read the data.

> **Note**
>
> You can choose how much data you want in snapshots and where to upload them. To do this, go to **Global Settings** > **Forensic Snapshots**. These options may not be available for all customers yet.

To create a snapshot, do as follows:

1. Go to a threat graph's **Analyze** tab.

   Alternatively, on the details page of the device, open the **Status** tab.
2. Click **Create forensic snapshot**.
3. Follow the steps in Upload a forensic snapshot to an AWS S3 bucket.

You can find the snapshots you generated in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Forensic Snapshots\`.

Snapshots generated from detections are in `%PROGRAMDATA%\Sophos\Endpoint Defense\Data\Saved Data\`.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/ThreatGraphs/ThreatAnalysisDetails/index.html.)

Forensic snapshots

Forensic snapshots get data from a Sophos log of a computer's activity so that you can do your own analysis.

You can create a forensic snapshot from a threat graph or from the **Status** tab in a device's details page.

Go to **Global Settings** > **Forensic Snapshots**.

You can configure how much data you want in your snapshots and where you want to put them.

> **Note**
>
> The configuration options may not be available for all customers yet.

Set the time period for the forensic snapshot

By default, a snapshot includes data for the previous two weeks.

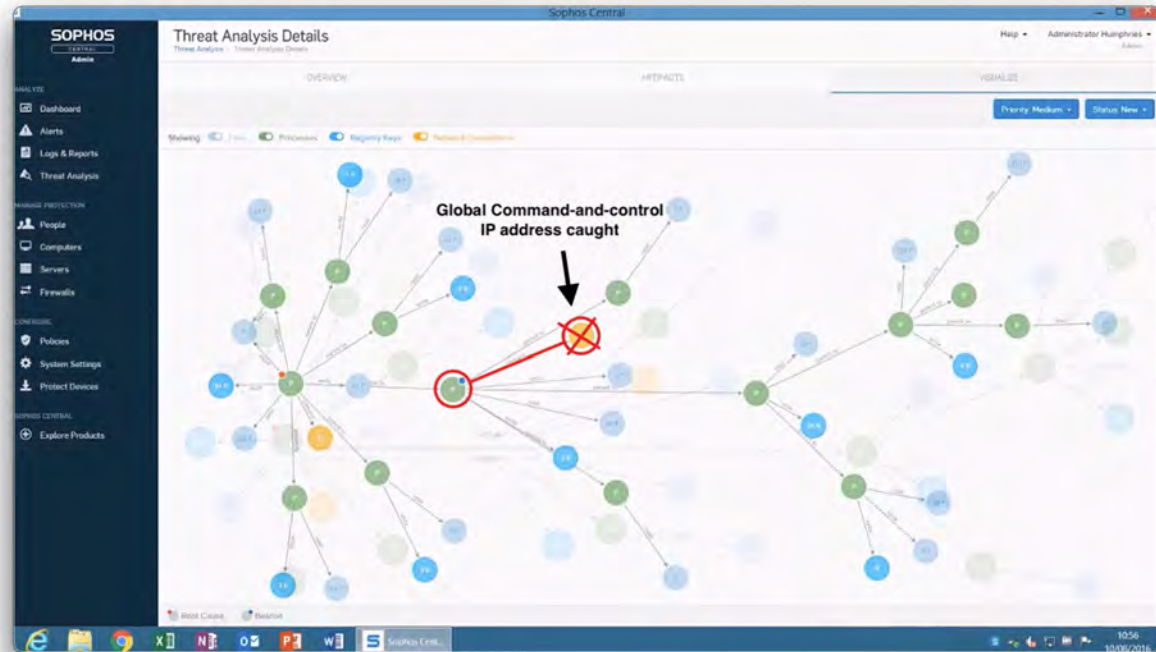
Here you can set a different time period or choose to include all the available data.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/GlobalSettings/ForensicSnapshots/index.html.)

522. The Accused Products also perform a method that includes *reporting the identity of the externally networked source to an externally networked pestware research entity, wherein the identity of the externally networked source is selected from a group consisting of an I.P. address, a URL, an email client and a program.*

523. For example, Intercept X's root cause analyses identify the IP addresses that originated each suspicious, or "beacon," event, and flag them as being "command-and-control" sites for malicious software if they are detected as being so by Sophos' threat analysis servers: "Here's the beacon event, the thing that was caught, in this case a fake salary report. Here's the root, Google Chrome. We see here that the fake report reached out to an IP address that we happen

to know to be a command-and-control site. This right here is the moment of conviction, when we discovered that what was executing was a piece of malware."



(*See* https://youtu.be/AOsjUjp4P7Q?t=48.)

524. Intercept X performs such root cause analyses and generates "detections based on data that users upload to the Sophos Data Lake."

Detections

Detections show you activity that you might need to investigate.

To see detections, go to **Threat Analysis Center** > **Detections**.

Detections identify activity on your devices that's unusual or suspicious but hasn't been blocked. They're different from events where we detect and block activity that we already know to be malicious.

We generate detections based on data that devices upload to the Sophos Data Lake.

We check that data against threat classification rules. When there's a match, we show a detection.

This page tells you how to use detections to look for potential threats.

> **Note**
>
> **Investigations** can automatically group related detections together for more advanced analysis. See Investigations.

(*See* https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/ThreatAnalysisCenter/Detections/index.html.)

525. Each claim in the ’932 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ‘932 Patent.

526. Defendant has been aware of the ’932 Patent since at least the date it was provided with a copy of this amended Complaint.

527. Defendant directly infringes at least claim 1 of the ’932 Patent, either literally or under the doctrine of equivalents, by performing the steps described above. For example, Defendant performs the claimed method as described above by running the Sophos security software and corresponding systems to protect its own computer and network operations. Defendant also performs the claimed method as described above when testing the operation of the Accused Products and corresponding systems. As another example, Defendant performs the claimed method as described above when providing or administering services to third parties,

customers, and partners using the Accused Products.

528. Defendant's partners, customers, and end users of the Accused Products and corresponding systems and services directly infringe at least claim 1 of the '932 Patent, literally or under the doctrine of equivalents, at least by using the Accused Products and corresponding systems and services, as described above.

529. Defendant has actively induced and is actively inducing infringement of at least claim 1 of the '932 Patent with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b). For example, Sophos encourages and induces customers with specific intent to use Sophos's security software in a manner that infringes claim 1 of the '932 Patent at least by offering and providing software that performs a method that infringes claim 1 when installed and operated by the customer, and by engaging in activities relating to selling, marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including the activities described below.

530. Defendant encourages, instructs, directs, and/or requires third parties—including its certified partners and/or customers to perform the claimed method using the software, services, and systems in infringing ways, as described above.

531. Defendant further encourages and induces its customers to infringe claim 1 of the '932 Patent: 1) by making its security services available on its website, providing applications that allow users to access those services, widely advertising those services, and providing technical support and instructions to users, and 2) through activities relating to marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including its Sophos security software, and services in the United States. (*See* https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx; WBR_SOP000525, https://partners.sophos.

com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1; WBR_SOP000209, https://secure2.sophos.com/en-us/security-news-trends/whitepapers/gated-wp/cybersecurity-system-buyers-guide.aspx%23formFrame.)

532. For example, Defendant shares instructions, guides, and manuals, which advertise and instruct third parties on how to use the software as described above, including at least customers and partners. (*Id.*) Defendant provides customer service and technical support to purchasers of the Accused Products and corresponding system and services, which directs and encourages customers to perform certain actions that use the Accused Products in an infringing manner. (*Id.*)

533. Defendant and/or its partners recommend and sell the Accused Products and provide technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. Each customer enters into a contractual relationship with Defendant and/or one of its partners, which obligates each customer to perform certain actions in order to use the Accused Products. Further, in order to receive the benefit of Defendant's or its partner's continued technical support and their specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that performs the claimed method and infringes the '932 Patent.

534. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Product remains operational for each customer through ongoing technical support. Defendant and/or Sophos's partners affirmatively aid and abet each customer's use of the Accused Products in a manner that performs the claimed method of, and infringes, the '932 Patent.

535. Defendant also contributes to the infringement of its partners, customers, and end-

users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the claimed methods, and not a staple article or commodity of commerce suitable for substantial non-infringing uses. Indeed, as shown above, the Accused Products and the example functionality described below have no substantial non-infringing uses but are specifically designed to practice the ’932 Patent.

536. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the functionality for identifying an origin of a malicious activity and related functionality, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. These processes are continually running when the system is in use and cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’932 Patent, that functionality could not be performed.

537. Additionally, the accused functionality, including the functionality for identifying an origin of a malicious activity and related functionality, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without extracting and analyzing a set of static features from an object and calibrating a feature vector on the basis of that analysis, Accused Products’ API call monitoring and other related functionality for identifying an origin of a malicious activity, the Accused Products could not deploy their malware detection and “root cause” features. These

processes are continually running when the system is in use and cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’932 Patent, that functionality could not be performed.

538. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Defendant constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionality constitute a material part of the inventions claimed at least because such an attack chain is integral to the processes identified above (such as “monitoring, using a kernel-mode driver, API call activity…”; “storing information related to the API call activity…”; “analyzing, heuristically, the API call activity…”; “identifying, based upon the API call activity, a suspected pestware object on the computer”; “identifying, in response to the identifying the suspected pestware object, a reference…”; and “reporting the identity of the externally networked source…”) as recited in the claims of the ’932 Patent. None of these products are staple goods—they are sophisticated and customized cyber security and malware detection products, methods, and systems.

539. Defendant’s infringing actions have continued after this amended Complaint was filed. Defendant had knowledge of the ’932 Patent and of the specific acts that constitutes infringement of the ’932 Patent at least on the date it was provided a copy of this amended Complaint, but has continued to engage in the infringing activity, including by selling, making, configuring, and installing the Accused Products and performing the accused functionality, and by engaging in activities that constituted infringement and contributory infringement as described

above.

540. The infringing actions of each partner, customer, and/or end-user of the Accused Products are attributable to Defendant. For example, Defendant directs and controls the activities or actions of its partners or others in connection with the Accused Products by contractual agreement or otherwise requiring partners or others to provide information and instructions to customers who acquire the Accused Products which, when followed, results in infringement. Defendant further directs and controls the operation of devices executing the Accused Products by programming the software which, when executed by a customer or end user, perform the method steps of at least claim 1 of the ’932 Patent.

541. Plaintiffs have suffered and continue to suffer damages, including lost profits, as a result of Defendant’s infringement of the ’932 Patent. Defendant is therefore liable to Plaintiffs under 35 U.S.C. § 284 for damages in an amount that adequately compensates Plaintiffs for Defendant’s infringement, but no less than a reasonable royalty.

542. Plaintiffs will continue to suffer irreparable harm unless this Court preliminarily and permanently enjoins Defendant, its agents, employees, representatives, and all others acting in concert with Defendant from infringing the ’932 Patent.

543. Defendant’s infringement of the ’932 Patent is knowing and willful. Defendant had actual knowledge of the ’932 Patent at least on the date it received a copy of this amended complaint, and had constructive knowledge of the ’932 Patent from at least the date Plaintiffs marked their products with the ’932 Patent and/or provided notice of the ’932 Patent on their website.

544. Despite Defendant’s knowledge of the Asserted Patents and Plaintiffs’ patented technology, Defendant made the deliberate decision to sell products and services that they knew

infringe the ’932 Patent. Defendant’s continued infringement of the ’932 Patent with knowledge of the ’932 Patent constitutes willful infringement.

<u>THIRTEENTH CAUSE OF ACTION</u>
(INFRINGEMENT OF THE ’123 PATENT)

545. Plaintiffs reallege and incorporate the preceding paragraphs of this Complaint.

546. Sophos has infringed and continues to infringe one or more claims of the ’123 Patent in violation of 35 U.S.C. § 271 in this judicial District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including Intercept X, when used for their ordinary and customary purposes, practice each element of at least claim 1 of the ’123 Patent as demonstrated below.

547. For example, claim 1 of the ’123 Patent recites:

> 1. A method comprising:
>
> receiving, at a base computer, details uniquely identifying one or more security products operating at a point in time on a remote computer;
>
> receiving, at the base computer, details uniquely identifying one or more security products operating on other remote computers in communication with the base computer;
>
> receiving, at the base computer, details of a process that has been executed by at least one of the other remote computers;
>
> determining, by the base computer and based on the received details of the process that has been executed by the least one of the other remote computers, that the process is a malware process not identified by the one or more security products operating on the at least one of the other remote computers; and
>
> determining, by the base computer, that the remote computer is vulnerable to the malware process, wherein the determination is based on the at least one of the other remote computers having a same or similar combination of security products as the combination of security products operating on the remote computer.

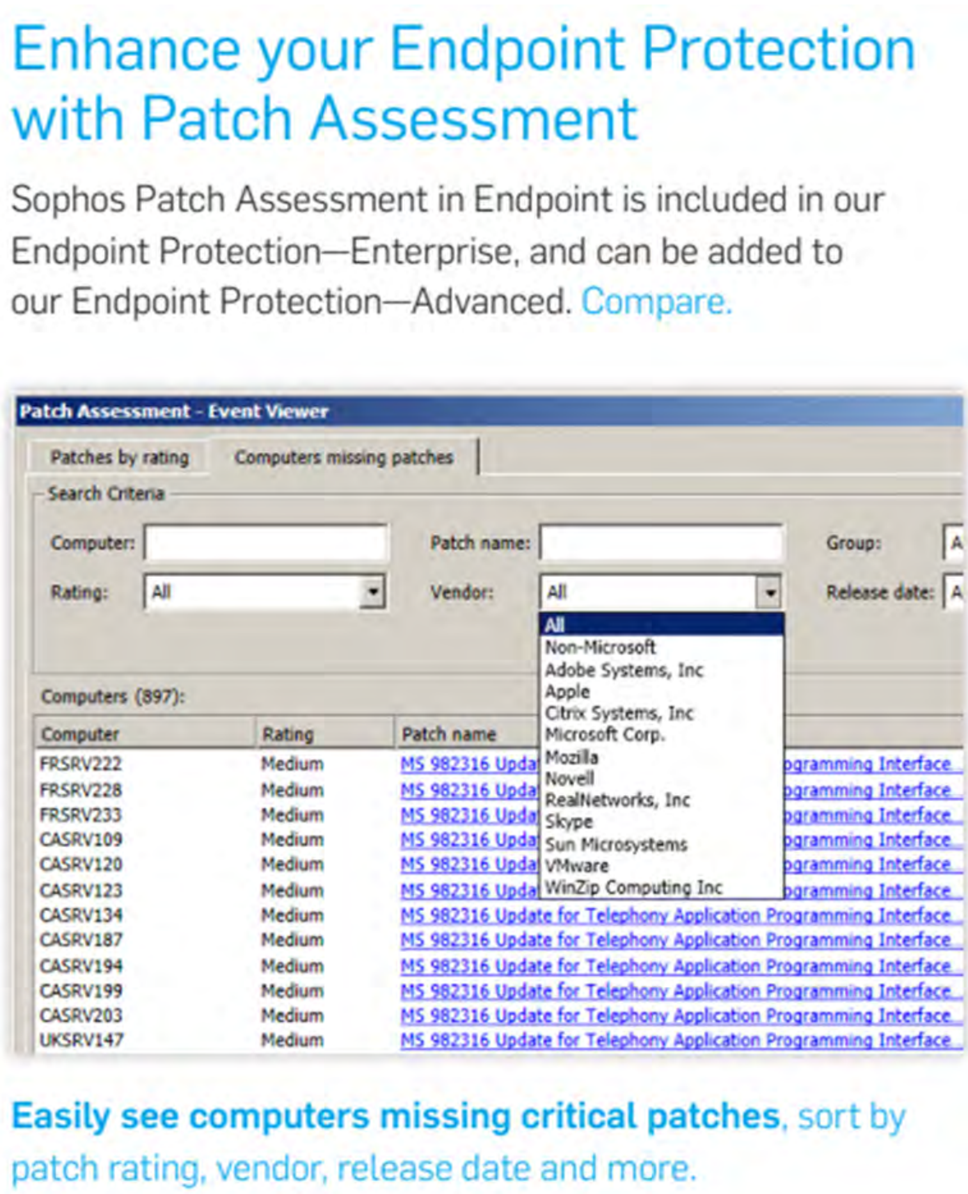
548. The Accused Products perform each of the method steps of claim 1 of the ’123 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform a

method, as further explained below.

549. The Accused Products perform a method that includes "*receiving, at a base computer, details uniquely identifying one or more security products operating at a point in time on a remote computer* and *receiving, at the base computer, details uniquely identifying one or more security products operating on other remote computers in communication with the base computer*. For example, "Sophos Patch Assessment in Endpoint is included in our Endpoint Protection—Enterprise, and can be added to our Endpoint Protection—Advanced." Indeed, Intercept X identifies, and sends to users of its "Sophos Enterprise Console," a list of "missing critical patches" from each endpoint that they oversee. Those details include the "Patch Name," the device which is missing that patch, the patch's release date, and the patch's vendor. Patches are "prioritized…by tying them to threats" and rated "critical, high, medium, and low" based on "security intelligence" collected from Intercept X endpoints and sent to Sophos' Cloud.

We prioritize patches by tying them to threats We prioritize patches by tying them to threats. SophosLabs rates patches—critical, high, medium and low—based on our security intelligence. We tell you which patches really matter so you can maximize protection with minimum effort.

* * * * *



(*See* https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophosendpointpatchassessmentdsna.pdf.)

550. The Accused Products perform a method that includes *receiving, at the base computer, details of a process that has been executed by at least one of the other remote computers*. For example, each endpoint on which Intercept X is installed sends data about the processes executing on it to Sophos' Cloud, which stores that data in a database and manages endpoints, which include remote computers, within a network. This data includes information about the

behavior of an object running on one or more remote computers. For example, data can be queried from each endpoint using "Live Discover" SQL queries through the "Threat Analysis Dashboard," to detect, for example, processes (running objects) that have made "[u]nusual changes to the registry" or to "search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere." Data about each process is automatically analyzed, marked as a "threat case" if appropriate, and displayed as such in the "Threat Analysis Center." Moreover, such data is also periodically uploaded by each endpoint to a cloud-based computer "Data Lake," which can be queried, for example, to obtain data about which processes executed on a given endpoint.

Live Discover

Live Discover allows you to check the devices that Sophos Central is managing, look for signs of a threat, or assess compliance.

You can use Live Discover queries to search devices for signs of threats that haven't been detected by other Sophos features. For example:

- Unusual changes to the registry.
- Failed authentications.
- A process running that is very rarely run.

You can also search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere, or if a user reports suspicious behavior on their device.

You can also check the compliance of each device. For example, you can search for out-of-date software or browsers with insecure settings.

This page tells you how to use Live Discover. You can also familiarize yourself with it by completing the Sophos XDR Training.

How queries work

We provide a range of queries for you to use to check your devices. You can use them as they are, or edit them (you'll need to be familiar with osquery or SQL). You can also create queries.

You can run queries to get information from different sources:

- Endpoint queries get the latest information from devices that are currently connected.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/learningContents/LiveDiscover.html; *see also* WBR_SOP000529, https://community.sophos.com/intercept-x-endpoint/b/blog/posts/introducing-the-new-threat-analysis-center.)

Data Lake queries

Data Lake queries let you search security and compliance data that your devices upload to the cloud.

You can run Data Lake queries with Live Discover, a feature in our Threat Analysis Center.

Live Discover now lets you choose which data source you use when you set up and run a query:

- Endpoints that are currently connected.
- The Data Lake in the cloud.

For help with Live Discover See Live Discover.

How the Data Lake works

We host the Data Lake and provide scheduled "hydration queries" that define which data your endpoints upload to it.

However, before you use Data Lake queries, you must make sure that data is being uploaded. To turn on uploads of data, See Data Lake uploads.

We store the data for 30 days.

We provide pre-prepared Data Lake queries you can run. You can use them as they are or edit them. You can also create your own queries.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/DataLakeQueries.html.)

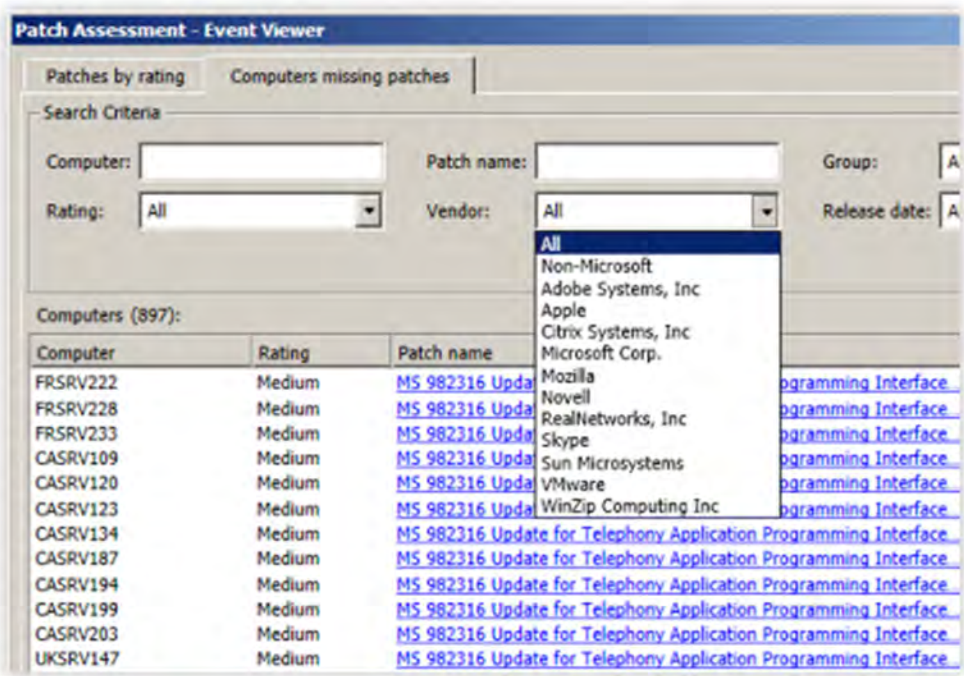
551. The Accused Products perform a method that includes *determining, by the base computer and based on the received details of the process that has been executed by the at least one of the other remote computers, that the process is a malware process not identified by the one or more security products operating on the at least one of the other remote computers*. For example, Intercept X identifies, and sends to users of its "Sophos Enterprise Console," a list of "missing critical patches" from each endpoint that they oversee. Those patches are "prioritized…by tying them to threats" and rated "critical, high, medium, and low" based on "security intelligence" collected from Intercept X endpoints.

We prioritize patches by tying them to threats We prioritize patches by tying them to threats. SophosLabs rates patches—critical, high, medium and low—based on our security intelligence. We tell you which patches really matter so you can maximize protection with minimum effort.

* * * * *

Enhance your Endpoint Protection with Patch Assessment

Sophos Patch Assessment in Endpoint is included in our Endpoint Protection—Enterprise, and can be added to our Endpoint Protection—Advanced. Compare.



Easily see computers missing critical patches, sort by patch rating, vendor, release date and more.

(*See* https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophosendpointpatchassessmentdsna.pdf.)

552. "[S]ecurity intelligence" is collected from Intercept X's endpoints via its "Threat Analysis" algorithms, which detect, for example, processes (running objects) that have made "[u]nusual changes to the registry" or to "search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere." Data about each process is automatically analyzed, marked as a "threat case" if appropriate, and displayed as such in the "Threat Analysis

Center." Moreover, such data is also periodically uploaded by each endpoint to a cloud-based computer "Data Lake," which can be queried, for example, to obtain data about which processes executed on a given endpoint.

Live Discover

Live Discover allows you to check the devices that Sophos Central is managing, look for signs of a threat, or assess compliance.

You can use Live Discover queries to search devices for signs of threats that haven't been detected by other Sophos features. For example:

- Unusual changes to the registry.
- Failed authentications.
- A process running that is very rarely run.

You can also search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere, or if a user reports suspicious behavior on their device.

You can also check the compliance of each device. For example, you can search for out-of-date software or browsers with insecure settings.

This page tells you how to use Live Discover. You can also familiarize yourself with it by completing the Sophos XDR Training.

How queries work

We provide a range of queries for you to use to check your devices. You can use them as they are, or edit them (you'll need to be familiar with osquery or SQL). You can also create queries.

You can run queries to get information from different sources:

- Endpoint queries get the latest information from devices that are currently connected.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/learningContents/LiveDiscover.html; *see also* WBR_SOP000529, https://community.sophos.com/intercept-x-endpoint/b/blog/posts/introducing-the-new-threat-analysis-center; https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophosendpointpatchassessmentdsna.pdf.)

Data Lake queries

Data Lake queries let you search security and compliance data that your devices upload to the cloud.

You can run Data Lake queries with Live Discover, a feature in our Threat Analysis Center.

Live Discover now lets you choose which data source you use when you set up and run a query:

- Endpoints that are currently connected.
- The Data Lake in the cloud.

For help with Live Discover See Live Discover.

How the Data Lake works

We host the Data Lake and provide scheduled "hydration queries" that define which data your endpoints upload to it.

However, before you use Data Lake queries, you must make sure that data is being uploaded. To turn on uploads of data, See Data Lake uploads.

We store the data for 30 days.

We provide pre-prepared Data Lake queries you can run. You can use them as they are or edit them. You can also create your own queries.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/DataLakeQueries.html.)

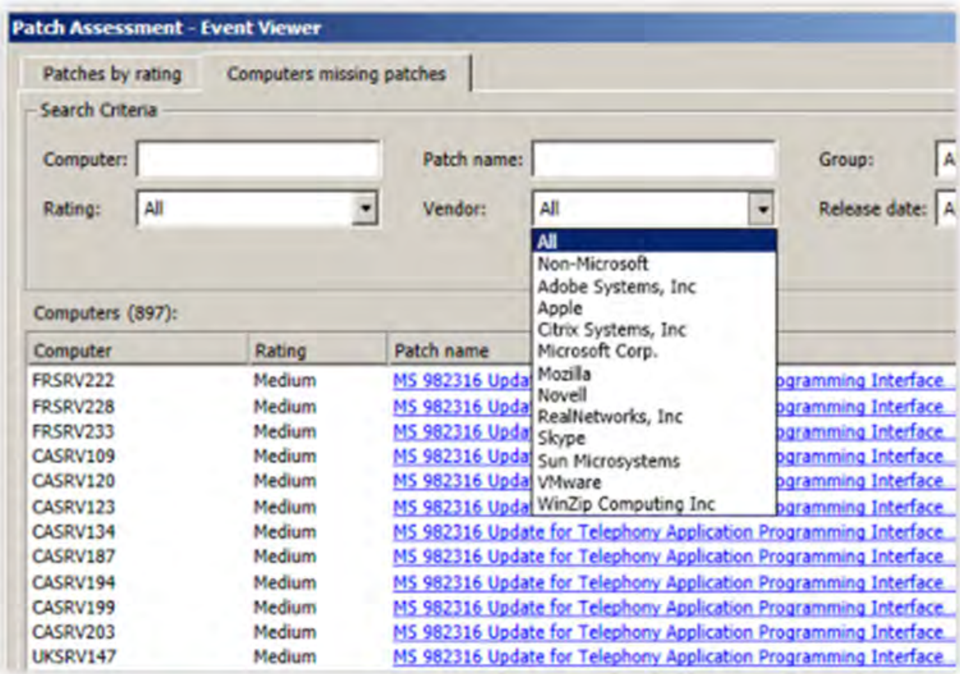
553. The Accused Products perform a method that includes *determining, by the base computer, that the remote computer is vulnerable to the malware process, wherein the determination is based on the at least one of the other remote computers having a same or similar combination of security products as the combination of security products operating on the remote computer*. For example, Intercept X identifies, and sends to users of its "Sophos Enterprise Console," a list of "missing critical patches" from each endpoint that they oversee. Those patches are "prioritized…by tying them to threats" and rated "critical, high, medium, and low" based on "security intelligence" collected from Intercept X endpoints.

We prioritize patches by tying them to threats We prioritize patches by tying them to threats. SophosLabs rates patches—critical, high, medium and low—based on our security intelligence. We tell you which patches really matter so you can maximize protection with minimum effort.

* * * * *

Enhance your Endpoint Protection with Patch Assessment

Sophos Patch Assessment in Endpoint is included in our Endpoint Protection—Enterprise, and can be added to our Endpoint Protection—Advanced. Compare.



Easily see computers missing critical patches, sort by patch rating, vendor, release date and more.

(*See* https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophosendpointpatchassessmentdsna.pdf.)

554. That "security intelligence" is collected from Intercept X's endpoints via its "Threat Analysis" algorithms, which detect, for example, processes (running objects) that have made "[u]nusual changes to the registry" or to "search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere." Data about each process is automatically analyzed, marked as a "threat case" if appropriate, and displayed as such in the "Threat Analysis

Center." Moreover, such data is also periodically uploaded by each endpoint to a cloud-based computer "Data Lake," which can be queried, for example, to obtain data about which processes executed on a given endpoint.

Live Discover

Live Discover allows you to check the devices that Sophos Central is managing, look for signs of a threat, or assess compliance.

You can use Live Discover queries to search devices for signs of threats that haven't been detected by other Sophos features. For example:

- Unusual changes to the registry.
- Failed authentications.
- A process running that is very rarely run.

You can also search devices for signs of a suspected or known threat if Sophos Central has found the threat elsewhere, or if a user reports suspicious behavior on their device.

You can also check the compliance of each device. For example, you can search for out-of-date software or browsers with insecure settings.

This page tells you how to use Live Discover. You can also familiarize yourself with it by completing the Sophos XDR Training.

How queries work

We provide a range of queries for you to use to check your devices. You can use them as they are, or edit them (you'll need to be familiar with osquery or SQL). You can also create queries.

You can run queries to get information from different sources:

- Endpoint queries get the latest information from devices that are currently connected.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/learningContents/LiveDiscover.html; *see also* WBR_SOP000529, https://community.sophos.com/intercept-x-endpoint/b/blog/posts/introducing-the-new-threat-analysis-center; https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophosendpointpatchassessmentdsna.pdf.)

Data Lake queries

Data Lake queries let you search security and compliance data that your devices upload to the cloud.

You can run Data Lake queries with Live Discover, a feature in our Threat Analysis Center.

Live Discover now lets you choose which data source you use when you set up and run a query:

- Endpoints that are currently connected.

- The Data Lake in the cloud.

For help with Live Discover See Live Discover.

How the Data Lake works

We host the Data Lake and provide scheduled "hydration queries" that define which data your endpoints upload to it.

However, before you use Data Lake queries, you must make sure that data is being uploaded. To turn on uploads of data, See Data Lake uploads.

We store the data for 30 days.

We provide pre-prepared Data Lake queries you can run. You can use them as they are or edit them. You can also create your own queries.

(*See* https://docs.sophos.com/central/Customer/help/en-us/central/Customer/concepts/DataLakeQueries.html.)

555. In another example, the Sophos "console policy allows scanning to be set to every 8 hours/24 hours/Week. If the period expires whilst an Endpoint is switched off, the scan will start the next time a machine is powered on." "Each endpoint checks the server for new patches before the start of each scan" and "[e]ndpoints only download from the server the patches relevant to their O/S and language." In addition, "[e]ach Endpoint uploads around 4 kBytes of results data to the server at the end of each scan."

How often do Endpoints scan for missing patches?

The console policy allows scanning to be set to every 8 hours/24 hours/Week. If the period expires whilst an Endpoint is switched off, the scan will start the next time a machine is powered on.

* * * * *

How often do Endpoints check for new patches?

Each endpoint checks the server for new patches before the start of each scan.

How do Endpoints determine which Patch files to download?

To optimize performance and minimize network traffic, Endpoints only download from the server the patches relevant to their O/S and language.

* * * * *

How much Patch data is uploaded to the Server from an Endpoint at the end of each scan?

Each Endpoint uploads around 4 kBytes of results data to the server at the end of each scan.

(*See* https://support.sophos.com/support/s/article/KB-000034181?language=en_US#q24.)

556. Each claim in the ’123 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ’123 Patent.

557. Defendant has been aware of the ’123 Patent since at least the date it received a copy of this amended Complaint. Further, Plaintiffs have marked its products with the ’123 Patent, including on its web site, since at least July 2020.

558. Defendant directly infringes at least claim 1 of the ’123 Patent, either literally or under the doctrine of equivalents, by performing the steps described above. For example, Defendant performs the claimed method as described above by running the Sophos security software and corresponding systems to protect its own computer and network operations. Defendant also performs the claimed method as described above when testing the operation of the Accused Products and corresponding systems. As another example, Defendant performs the claimed method as described above when providing or administering services to third parties, customers, and partners using the Accused Products.

559. Defendant’s partners, customers, and end users of the Accused Products and corresponding systems and services directly infringe at least claim 1 of the ’123 Patent, literally or under the doctrine of equivalents, at least by using the Accused Products and corresponding systems and services, as described above.

560. Defendant has actively induced and is actively inducing infringement of at least claim 1 of the ’123 Patent with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b). For example,

Sophos encourages and induces customers with specific intent to use Sophos's security software in a manner that infringes claim 1 of the '123 Patent at least by offering and providing software that performs a method that infringes claim 1 when installed and operated by the customer, and by engaging in activities relating to selling, marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including the activities described below.

561. Defendant encourages, instructs, directs, and/or requires third parties—including its certified partners and/or customers to perform the claimed method using the software, services, and systems in infringing ways, as described above.

562. Defendant further encourages and induces its customers to infringe claim 1 of the '123 Patent: 1) by making its security services available on its website, providing applications that allow users to access those services, widely advertising those services, and providing technical support and instructions to users, and 2) through activities relating to marketing, advertising, promotion, installation, support, and distribution of the Accused Products, including its Sophos security software, and services in the United States. (*See* https://www.sophos.com/en-us/products/endpoint-antivirus/how-to-buy.aspx; WBR_SOP000525, https://partners.sophos.com/english/directory/search?lat=30.267153&lng=-97.7430608&dMI=100&p=1; WBR_SOP000209, https://secure2.sophos.com/en-us/security-news-trends/whitepapers/gated-wp/cybersecurity-system-buyers-guide.aspx%23formFrame.)

563. For example, Defendant shares instructions, guides, and manuals, which advertise and instruct third parties on how to use the software as described above, including at least customers and partners. (*Id.*) Defendant provides customer service and technical support to purchasers of the Accused Products and corresponding system and services, which directs and encourages customers to perform certain actions that use the Accused Products in an infringing

manner. (*Id.*)

564. Defendant and/or its partners recommend and sell the Accused Products and provide technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. Each customer enters into a contractual relationship with Defendant and/or one of its partners, which obligates each customer to perform certain actions in order to use the Accused Products. Further, in order to receive the benefit of Defendant's or its partner's continued technical support and their specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that performs the claimed method and infringes the '932 Patent.

565. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Product remains operational for each customer through ongoing technical support. Defendant and/or Sophos's partners affirmatively aid and abet each customer's use of the Accused Products in a manner that performs the claimed method of, and infringes, the '123 Patent.

566. Defendant also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the claimed methods, and not a staple article or commodity of commerce suitable for substantial non-infringing uses. Indeed, as shown above, the Accused Products and the example functionality described below have no substantial non-infringing uses but are specifically designed to practice the '123 Patent.

567. Indeed, as shown above, the Accused Products have no substantial non-infringing

uses because the accused functionality, including the process and vulnerability analysis and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. These processes are continually running when the system is in use and cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’123 Patent, that functionality could not be performed.

568. Additionally, the accused functionality, including the process and vulnerability analysis and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without compiling and analyzing data about the behavior of an object running on one or more remote computers, the Accused Products could not detect processes (running objects) that have made unusual changes to the registry or to search devices for signs of a suspected or known threat. These processes are continually running when the system is in use and cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice claimed in the ’123 Patent, that functionality could not be performed.

569. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Defendants constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionalities (including the process

and vulnerability analysis functionality) constitute a material part of the inventions claimed because such analysis is integral to the processes identified herein (such as "*determining, by the base computer, that the remote computer is vulnerable to the malware process*") as recited in the claims of the '123 Patent. None of these products are staple goods—they are sophisticated and customized cyber security and malware detection products, methods, and systems.

570. Defendant's infringing actions have continued after it received a copy of this amended complaint. Defendant had knowledge of the '123 Patent and of the specific acts that constitutes infringement of the '123 Patent at least based on the original Complaint, but has continued to engage in the infringing activity, including by selling, making, configuring, and installing the Accused Products and performing the accused functionality, and by engaging in activities that constituted infringement and contributory infringement as described above.

571. The infringing actions of each partner, customer, and/or end-user of the Accused Products are attributable to Defendant. For example, Defendant directs and controls the activities or actions of its partners or others in connection with the Accused Products by contractual agreement or otherwise requiring partners or others to provide information and instructions to customers who acquire the Accused Products which, when followed, results in infringement. Defendant further directs and controls the operation of devices executing the Accused Products by programming the software which, when executed by a customer or end user, perform the method steps of at least claim 1 of the '123 Patent.

572. Plaintiffs have suffered and continue to suffer damages, including lost profits, as a result of Defendant's infringement of the '123 Patent. Defendant is therefore liable to Plaintiffs under 35 U.S.C. § 284 for damages in an amount that adequately compensates Plaintiffs for Defendant's infringement, but no less than a reasonable royalty.

573. Plaintiffs will continue to suffer irreparable harm unless this Court preliminarily and permanently enjoins Defendant, its agents, employees, representatives, and all others acting in concert with Defendant from infringing the ’123 Patent.

574. Defendant’s infringement of the ’123 Patent is knowing and willful. Defendant had actual knowledge of the ’123 Patent at least by the date it received a copy of this amended complaint, and had constructive knowledge of the ’123 Patent from at least the date Plaintiffs marked their products with the ’123 Patent and/or provided notice of the ’123 Patent on their website.

PRAYER FOR RELIEF

WHEREFORE, Plaintiffs respectfully request the following relief:

a) That this Court adjudge and decree that Defendant has been, and are currently, infringing each of the Asserted Patents;

b) That this Court award damages to Plaintiffs to compensate it for Defendant’s past infringement of the Asserted Patents, through the date of trial in this action;

c) That this Court award pre- and post-judgment interest on such damages to Plaintiffs;

d) That this Court order an accounting of damages incurred by Plaintiffs from six years prior to the date this lawsuit was filed through the entry of a final, non-appealable judgment;

e) That this Court determine that this patent infringement case is exceptional pursuant to 35 U.S.C. §§ 284 and 285 and award Plaintiffs its costs and attorneys’ fees incurred in this action;

f) That this Court award increased damages under 35 U.S.C. § 284;

g) That this Court preliminarily and permanently enjoin Defendant from infringing any of the Asserted Patents;

h) That this Court order Defendant to:

(i) recall and collect from all persons and entities that have purchased any and all products found to infringe any of the Asserted Patents that were made, offered for sale, sold, or otherwise distributed in the United States by Defendant or anyone acting on its behalf;

(ii) destroy or deliver all such infringing products to Plaintiffs;

(iii) revoke all licenses to all such infringing products;

(iv) disable all web pages offering or advertising all such infringing products;

(v) destroy all other marketing materials relating to all such infringing products;

(vi) disable all applications providing access to all such infringing software; and

(vii) destroy all infringing software that exists on hosted systems,

i) That this Court, if it declines to enjoin Defendant from infringing any of the Asserted Patents, award damages for future infringement in lieu of an injunction; and

j) That this Court award such other relief as the Court deems just and proper.

DEMAND FOR JURY TRIAL

OpenText respectfully requests a trial by jury on all issues triable thereby.

DATED: October 4, 2022

By:*/s/ Jeffrey D. Mills*
Jeffrey D. Mills
Texas Bar No. 24034203
KING & SPALDING LLP
500 West Second St.
Suite 1800
Austin, Texas 78701
Telephone: (512) 457-2027
Facsimile: (512) 457-2100
jmills@kslaw.com

Mark D. Siegmund
STECKLER WAYNE
CHERRY & LOVE, PLLC
8416 Old McGregor Rd.
Waco, Texas 76712
Telephone: (254) 651-3690
Facsimile: (254) 651-3689
mark@swclaw.com

Steve Sprinkle
Texas Bar No. 00794962
SPRINKLE IP LAW GROUP, P.C.
1301 W. 25th Street, Suite 408
Austin, Texas 78705
Telephone: 512-637-9220
ssprinkle@sprinklelaw.com

Christopher C. Campbell (DC Bar No. 444262)
Patrick M. Lafferty *(pro hac vice)*
KING & SPALDING LLP
1700 Pennsylvania Avenue, NW
Suite 200
Washington, DC 20006
Telephone: (202) 626-5578
Facsimile: (202) 626-3737
ccampbell@kslaw.com
plafferty@kslaw.com

Britton F. Davis *(pro hac vice)*
Brian Eutermoser (*pro hac vice)*
KING & SPALDING LLP
1401 Lawrence Street
Suite 1900
Denver, CO 80202
Telephone: (720) 535-2300
Facsimile: (720) 535-2400
bfdavis@kslaw.com
beutermoser@kslaw.com

Attorneys for Plaintiffs Open Text, Inc. and Webroot, Inc.

CERTIFICATE OF SERVICE

A true and correct copy of the foregoing instrument was served or delivered electronically via the U.S. District Court ECF filing system to all counsel of record on this 4th day of October, 2022.

/s/ Jeffrey D. Mills
Jeffrey D. Mills

Exhibit 1

U 8164950

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

October 07, 2021

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THIS OFFICE OF:

U.S. PATENT: *8,418,250*
ISSUE DATE: *April 09, 2013*

By Authority of the
Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office

R GLOVER
Certifying Officer

US008418250B2

(12) **United States Patent**
Morris et al.

(10) **Patent No.:** **US 8,418,250 B2**
(45) **Date of Patent:** **Apr. 9, 2013**

(54) **METHODS AND APPARATUS FOR DEALING WITH MALWARE**

(75) Inventors: **Melvyn Morris**, Turnditch (GB); **Paul Stubbs**, Wyboston (GB); **Markus Hartwig**, Milton Keynes (GB); **Darren Harter**, Hucclecote (GB)

(73) Assignee: **Prevx Limited** (GB)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1111 days.

(21) Appl. No.: **11/477,807**

(22) Filed: **Jun. 30, 2006**

(65) **Prior Publication Data**

US 2007/0016953 A1 Jan. 18, 2007

(30) **Foreign Application Priority Data**

Jun. 30, 2005 (GB) 0513375.6

(51) **Int. Cl.**
G06F 21/00 (2006.01)
(52) **U.S. Cl.** **726/24**
(58) **Field of Classification Search** 726/24
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,338,141 B1	1/2002	Wells	
6,772,346 B1	8/2004	Chess	
6,944,772 B2	9/2005	Dozortsev	
7,013,483 B2	3/2006	Cohen et al.	
7,093,239 B1	8/2006	van der Made	
2001/0052014 A1*	12/2001	Sheymov et al.	709/225
2002/0099952 A1*	7/2002	Lambert et al.	713/200
2003/0023857 A1	1/2003	Hinchlife	
2003/0084323 A1*	5/2003	Gales	713/200
2003/0101381 A1	5/2003	Mateev	
2003/0135791 A1	7/2003	Natvig	
2003/0177394 A1*	9/2003	Dozortsev	713/201
2004/0006704 A1	1/2004	Dahlstrom	
2004/0039921 A1	2/2004	Chuang	
2004/0068618 A1	4/2004	Hooker	
2004/0083408 A1	4/2004	Spiegel	

(Continued)

FOREIGN PATENT DOCUMENTS

EP	1 280 040	1/2003
EP	1 315 066	5/2003

(Continued)

OTHER PUBLICATIONS

U.S Appl. No. 60/789,156.

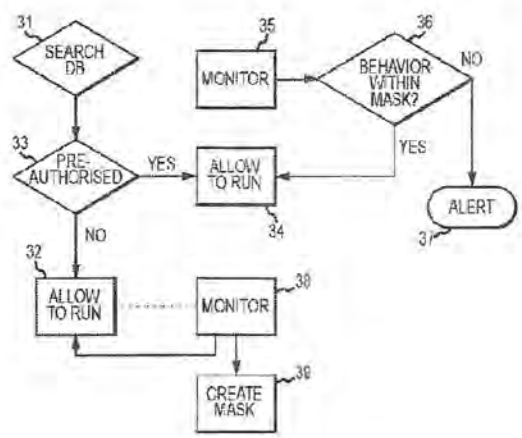
(Continued)

Primary Examiner — Cordelia Zecher
(74) *Attorney, Agent, or Firm* — Sheridan Ross P.C.

(57) **ABSTRACT**

In one aspect, a method of classifying a computer object as malware includes receiving at a base computer data about a computer object from each of plural remote computers on which the object or similar objects are stored. The data about the computer object received from the plural computers is compared in the base computer. The computer object is classified as malware on the basis of said comparison. In one embodiment, the data about the computer object includes one or more of: executable instructions contained within or constituted by the object; the size of the object; the name of the object; the logical storage location or path of the object on the respective remote computers; the vendor of the object; the software product and version associated with the object; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

30 Claims, 3 Drawing Sheets



Copy provided by USPTO from the PIRS Image Database on 10-05-2021

U.S. PATENT DOCUMENTS

2004/0083810	A1	5/2004	Racine	
2004/0153644	A1	8/2004	McCorkendale	
2004/0255165	A1	12/2004	Szor	
2005/0021994	A1 *	1/2005	Barton et al.	713/200
2005/0022014	A1	1/2005	Shipman	
2005/0027686	A1	2/2005	Shipp	
2005/0086500	A1	4/2005	Albornoz	
2005/0210035	A1 *	9/2005	Kester et al.	707/10
2008/0209562	A1	8/2008	Szor	
2008/0320595	A1	12/2008	van der Made	
2009/0271867	A1	10/2009	Zhang	

FOREIGN PATENT DOCUMENTS

EP	1 549 012	12/2003
EP	1 280 040 A3	3/2004
EP	1 549 012	6/2005
EP	1 549 012 A1	6/2005
EP	1536341 A2	6/2005
JP	3-233629	10/1991
JP	6-110718	4/1994
JP	9-504395	4/1997
JP	2003-196112	7/2003
WO	WO 95/12162	5/1995
WO	WO 96/30829	10/1996
WO	WO 99/15966	4/1999
WO	02/33525	4/2002
WO	WO 02/33525 A2	4/2002
WO	03/021402	3/2003
WO	WO 03/021402 A2	3/2003
WO	2004/097602	11/2004
WO	WO 2004/097602 A2	11/2004

OTHER PUBLICATIONS

U.S. Appl 11/694,261.

"Ghostbuster 3", Mihai Chirac, pp. 1-7, Prevx1 Sep. 2005.

A. Stephan; "Defeating Polymorphism: Beyond Emuliation"; Virus Bulletin Conference; Oct. 2005.

Japanese Office Action issued in JP 2008-518975 on Sep. 13, 2011.

English Language Translation of Japanese Office Action issued in JP 2008-518975 on Sep. 13, 2011.

English Language Abstract and Translation of JP 6-110718 published Apr. 22, 1994.

English Language Abstract of JP 3-233629 published Oct. 17, 1991.

English Language Abstract of JP 2003-196112 published Jul. 11, 2003.

International Search Report issued in PCT/GB2006/002439 issued Jan. 15, 2007.

Zenkin, D., "Fighting Against the Invisible Enemy—Methods for detecting an unknown virus" Computers & Security, Elsevier Science Publishers; Amsterdam, NL; vol. 20; No. 4; Jul. 31, 2001; pp. 316-321, XP004254268.

* cited by examiner

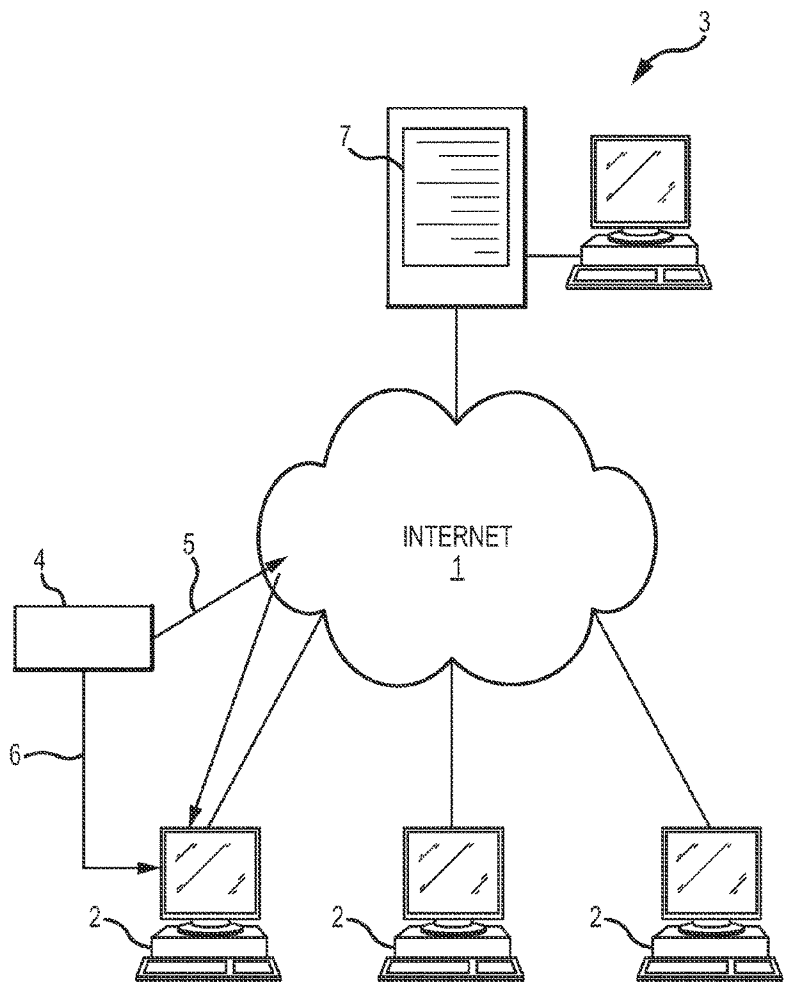


FIG.1

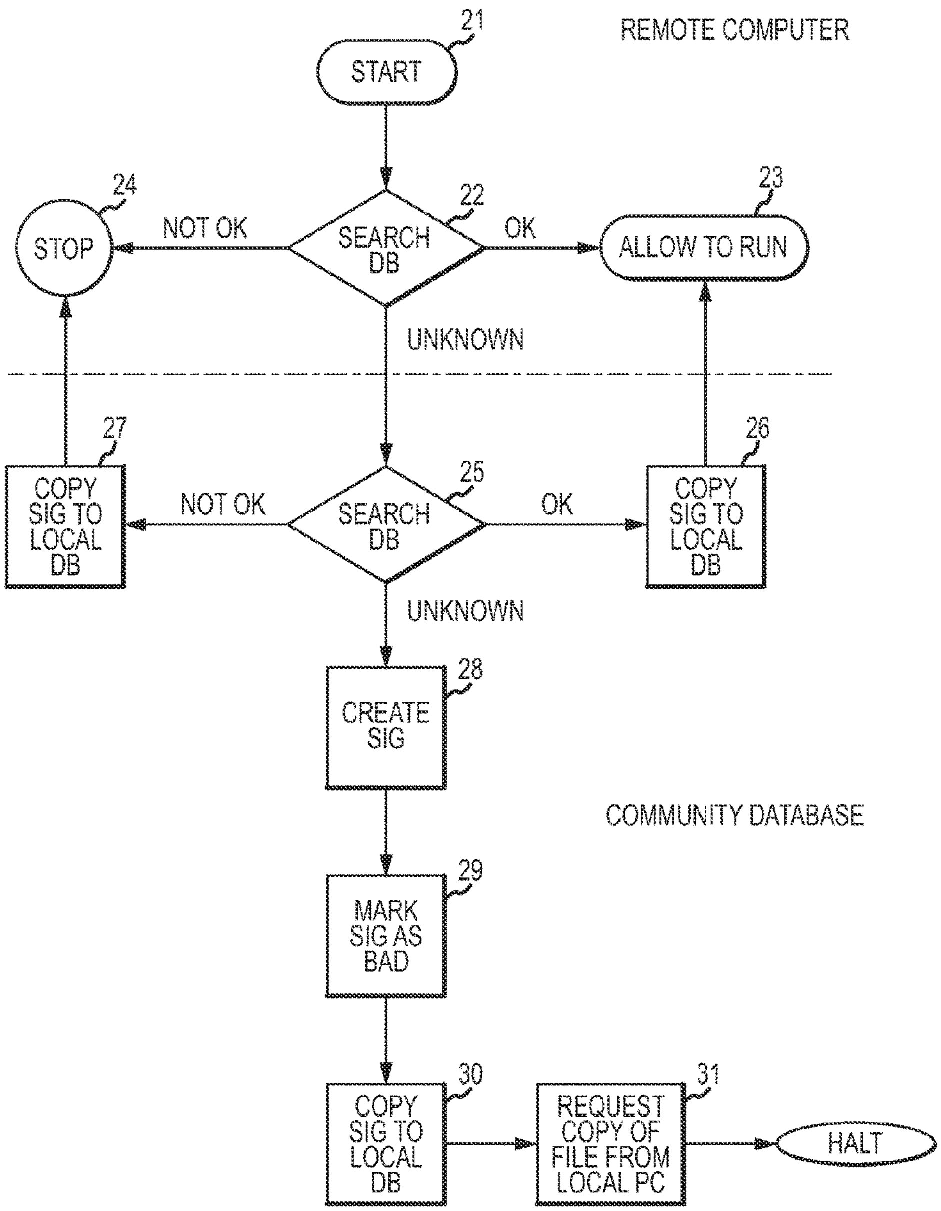

21
REMOTE COMPUTER
START
24
STOP
NOT OK
22
SEARCH DB
OK
23
ALLOW TO RUN
UNKNOWN
27
COPY SIG TO LOCAL DB
NOT OK
25
SEARCH DB
OK
26
COPY SIG TO LOCAL DB
UNKNOWN
28
CREATE SIG
COMMUNITY DATABASE
29
MARK SIG AS BAD
30
COPY SIG TO LOCAL DB
31
REQUEST COPY OF FILE FROM LOCAL PC
HALT

FIG.2

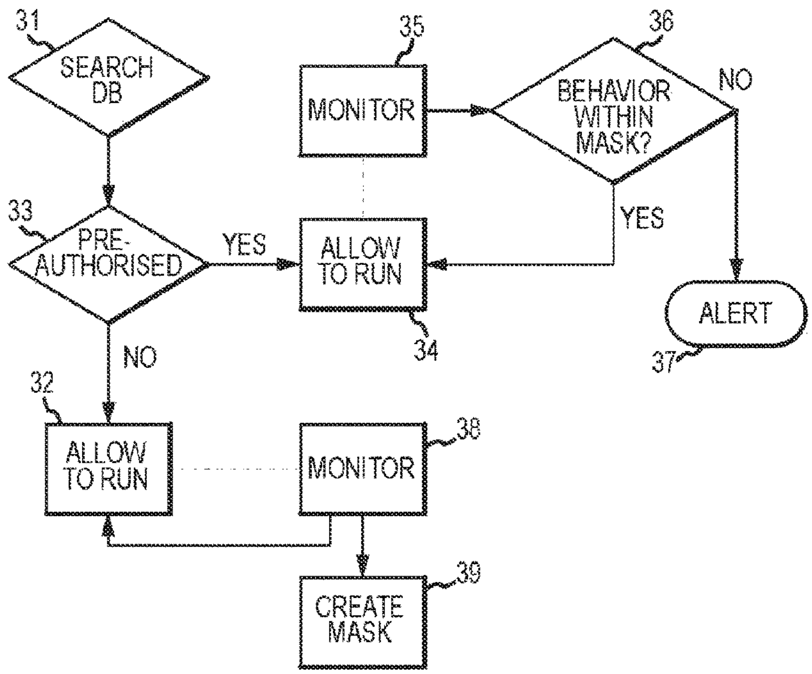

31
SEARCH DB
33
PRE-AUTHORISED
YES
NO
32
ALLOW TO RUN
35
MONITOR
36
BEHAVIOR WITHIN MASK?
NO
YES
ALLOW TO RUN
34
ALERT
37
MONITOR
38
CREATE MASK
39

FIG.3

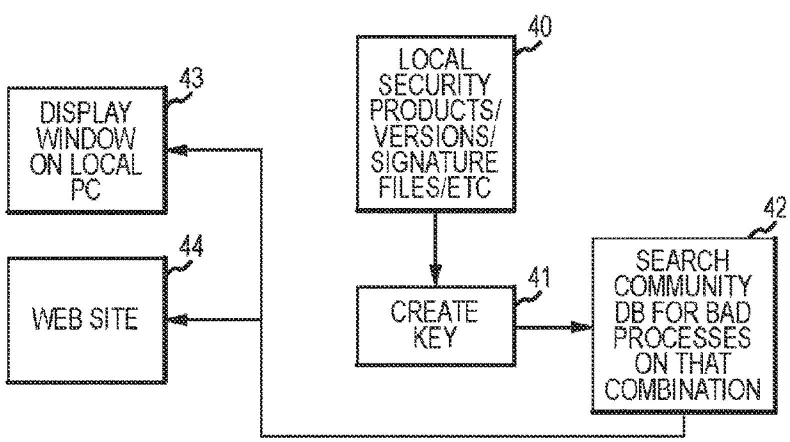

43
DISPLAY WINDOW ON LOCAL PC
44
WEB SITE
40
LOCAL SECURITY PRODUCTS/ VERSIONS/ SIGNATURE FILES/ETC
41
CREATE KEY
42
SEARCH COMMUNITY DB FOR BAD PROCESSES ON THAT COMBINATION

FIG.4

METHODS AND APPARATUS FOR DEALING WITH MALWARE

The present invention relates generally to methods and apparatus for dealing with malware. In one aspect, the present invention relates to a method and apparatus for classifying a computer object as malware. In another aspect, the present invention relates to a method and apparatus for determining the protection that a remote computer has from malware. In another aspect, the present invention relates to a method and apparatus for classifying a computer object as malware or as safe. In another aspect, the present invention relates to a method of installing software on a computer.

The term “malware” is used herein to refer generally to any executable computer file or, more generally “object”, that is or contains malicious code, and thus includes viruses, Trojans, worms, spyware, adware, etc. and the like.

A typical anti-malware product, such as anti-virus scanning software, scans objects or the results of an algorithm applied to the object or part thereof to look for signatures in the object that are known to be indicative of the presence of a virus. Generally, the method of dealing with malware is that when new types of malware are released, for example via the Internet, these are eventually detected. Once new items of malware have been detected, then the service providers in the field generate signatures that attempt to deal with these and these signatures are then released as updates to their anti-malware programs. Heuristic methods have also been employed.

These systems work well for protecting against known malicious objects. However, since they rely on signature files being generated and/or updated, there is inevitably a delay between a new piece of malware coming into existence or being released and the signatures for combating that malware being generated or updated and supplied to users. Thus, users are at risk from new malware for a certain period of time which might be up to a week or even more. Moreover, in order to try to defeat anti-virus products, the malware writers use obfuscation techniques in order to attempt to hide the signature or signature base data of the virus code from detection. Typically, the obfuscation involves encrypting or packing the viral code.

WO-A-2004/097602 describes a system that analyses computer files received or generated by a local computer and compares these with a database of known files to determine whether a particular file is known and if so whether it has been known about long enough that it can be regarded as “safe”. However, in practice, on its own this is not likely to provide for adequate protection because, for example, the active payload of a virus or Trojan may only be programmed to activate at a particular date, or upon receiving a message or instruction from a local or remote system or process, or on the occurrence of a particular event that may be many months or even years after the process has been first run or is released. Thus, just looking at the age of a file is an unsatisfactory way of determining whether it is properly safe and will remain so.

In the system of US-A-2004/0083408, a worm in a file is detected by examining connection attempts made by the specific file running on a computer.

U.S. Pat. No. 6,944,772, U.S. Pat. No. 6,772,346, EP-A-1549012 and EP-A-1280040 all disclose “community-based” anti-malware systems in which a plurality of “local” computers all connect via a network (which may be a LAN or the Internet, for example) to a central computer. On encountering a file that is not already known to them, the local computers send a request to the central computer for authorisation to run the file. If the file is recognised at the central

computer, then the central computer can send permission for the local computer to run the file if the file is known to be safe or send a “deny” command if the file is known to be malicious. However, in each of these prior art proposals, if the file is not known at the central computer, then the whole file is sent to the central computer where it can be analysed to determine whether it should be regarded as safe or malware. Such analysis is typically carried out manually or “semi-manually” by subjecting the file to detailed analysis, for example by emulation or interpretation, which can still take days given the human involvement that is typically required. There is therefore still a considerable period of time before a new file is classified as safe or as malware. In the case of these prior art systems, the request for authorisation to run the file that is sent by a local computer to the central computer may comprise sending a checksum or “signature” or “key” that uniquely represents the file.

A similar community-based anti-malware system is disclosed in WO-A-02/33525. In this system, in the case that a local computer is seeking clearance to run a file that is not known by the central computer to be safe or malware, some limited audit information about the prevalence of the file on other local computers can be sent to a human system administrator associated with the local computer that is seeking permission to run the file. The human system administrator can therefore make a better informed though still “manual” decision as to whether or not the file is safe to run.

In the system of US-A-2004/0073810, a metafile containing data about an attachment or other transmitted file is sent to a central computer. The data about that file is analysed to determine a likelihood of the transmitted file being malware. A specific example given is that if the transmitted file has been transmitted via at least a certain number of servers, then it should be treated as malware.

In the systems disclosed in US-A-2005/0021994 and US-A-2004/0153644, pre-approved files, which may be certified as safe by for example the software vendor associated with the files, may be permitted always to run without further checking. In one embodiment of the system of US-A-2004/0153644, monitoring is carried out to decide that a file is malicious if an abnormally high number of requests by that file is received at a central authority from plural local computers in a time period or if an abnormally high number of requests by that file on a single computer is received from the single local computer in a time period.

In the system of US-A-2004/0006704, a comparison is made between installed versions of software on a computer with a database of software versions and their known vulnerabilities. A user of the computer can therefore be informed of specific risks and how to minimise those risks by updating existing or installing new software.

In the system of WO-A-03/021402, a central database holds a virtual image of all files stored on each of plural local computers. If a threat in one local computer is identified, other local computers with a similar configuration can be notified of the risk.

Thus, the prior art systems either rely on deep analysis of a new object in order to determine whether or not the object is malicious, which introduces delay and therefore risk to users during the period that the file is analysed and new anti-malware signatures distributed, or limited analysis of the operation of the particular object or its method of transmission to a computer is carried out to decide a likelihood of the object being malicious.

According to a first aspect of the present invention, there is provided a method of classifying a computer object as malware, the method comprising:

at a base computer, receiving data about a computer object from each of plural remote computers on which the object or similar objects are stored;

comparing in the base computer the data about the computer object received from the plural computers; and,

classifying the computer object as malware on the basis of said comparison.

Compared to the prior art that relies solely on signature matching, this aspect allows a comparison to be made between the objects and/or their effects on the different remote computers to determine whether or not a particular object should be classed as good or as malware. Sophisticated pattern analysis can be carried out. This allows a rapid determination of the nature of the object to be made, without requiring detailed analysis of the object itself as such to determine whether it malware and also avoids the need to generate new signatures to be used for signature matching as in the conventional prior art anti-virus software.

In a preferred embodiment, the data about the computer object that is sent from the plural remote computers to the base computer and that is used in the comparison includes one or more of: executable instructions contained within or constituted by the object; the size of the object; the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

Preferably, the comparing identifies relationships between the object and other objects. In an example, this can be used immediately and automatically to mark a child object as bad (or good) if the or a parent or other related object is bad (or good). Thus, if at least one other object to which said object is related is classed as malware, then the method may comprise classifying said object as malware. Said other objects include the object or similar objects stored on at least some of the remote computers. Said other objects may include other objects that are parent objects or child objects or otherwise process-related objects to said object.

In a most preferred embodiment, the data is sent in the form of key that is obtained by a hashing process carried out in respect of the objects on the respective remote computers. A major advantage of using such a key is that it keeps down the volume of data that needs to be transmitted to the base computer. Given that there may be thousands or even millions of connected remote computers and further given that each may send details about very many objects, this can be an important advantage.

The key preferably has at least one component that represents executable instructions contained within or constituted by the object. This important preferred feature allows a comparison to be made at the base computer of only the executable instructions of the object. This means for example that differently named objects that basically have the same executable instructions, which is often an indicator that the objects are malware, can nevertheless be regarded as the "same" object for this purpose. As another example, a new version of a program may be released which has minor changes compared to a previous version already known to the base computer and which in substance, at least in respect of the executable instructions, can be regarded as being the same as the previous version. In that case, the minor differences can be ignored and the objects regarded as being the same. Not only is this useful in distinguishing between malware and for example revised versions of previous software, it also keeps down the data transmission and storage requirements because the base computer can inform the remote computers that an apparently new object is for this purpose the same as a previously known object, thus avoiding having the remote computers send full details about the object or the object itself to the base computer.

The key preferably has at least one component that represents data about said object. Said data about said object may include at least one of: the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

The key preferably has at least one component that represents the physical size of the object.

Where more than one of these components are present in the key, the plural components are preferably severable.

The method may comprise initially classifying an object as not malware, generating a mask for said object that defines acceptable behaviour for the object, and comprising monitoring operation of the object on at least one of the remote computers and reclassifying the object as malware if the actual monitored behaviour extends beyond that permitted by the mask. This provides an efficient and effective way of monitoring the behaviour of an object that has been classified or regarded as good and allows the object to be reclassified quickly as malware if the behaviour of the object warrants it.

According to a second aspect of the present invention, there is provided apparatus for classifying a computer object as malware, the apparatus comprising:

a base computer constructed and arranged to receive data about a computer object from each of plural remote computers on which the object or similar objects are stored;

the base computer being constructed and arranged to compare the data about the computer object received from said plural computers; and,

the base computer being constructed and arranged to classify the computer object as malware on the basis of said comparison.

According to a third aspect of the present invention, there is provided a method of providing data about a computer object from a remote computer to a base computer so that a comparison can be made at the base computer with similar data received from other remote computers, the method comprising:

providing from a remote computer to a base computer data about a computer object that is stored on the remote computer;

the data including one or more of: executable instructions contained within or constituted by the object; the size of the object; the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers;

the data being sent in the form of key that is obtained by a hashing process carried out in respect of the object on the remote computer.

This method, which may be carried out by so-called agent software running on the remote computer, allows for efficient sending of data to the base computer, which minimises data

transmission and storage requirements and also permits rapid analysis to be made at the base computer.

The key preferably has at least one component that represents executable instructions contained within or constituted by the object.

The key preferably has at least one component that represents data about said object. Said data about said object may include at least one of: the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

The key preferably has at least one component that represents the physical size of the object.

According to a fourth aspect of the present invention, there is provided a method of determining the protection that a remote computer has from malware, the method comprising:

receiving at a base computer details of all or selected security products operating at a point in time on said remote computer;

receiving similar information from other remote computers connected to the base computer; and,

identifying any malware processes that were not identified by said other remote computers having that particular combination of security products.

In this way, the base computer can be used to obtain information as to whether for example a particular, specific combination of operating system and various security products, including settings and signature files existing at a point in time, renders a particular computer having those products and settings susceptible or vulnerable to any particular malware object. The user can be advised accordingly and for example provided with recommendations for remedying the situation.

The method may therefore comprise providing information to the user of said remote computer that said remote computer may be susceptible to attack by said malware processes on the basis of said identifying.

The details of all or selected security products preferably includes the name of the security products, versions, and loaded signature files.

According to a fifth aspect of the present invention, there is provided apparatus for determining the protection that a remote computer has from malware, the apparatus comprising:

a base computer constructed and arranged to receive computer details of all or selected security products operating at a point in time on said remote computer;

the base computer being constructed and arranged to receive similar information from other remote computers connected to the base computer; and,

the base computer being constructed and arranged to identify any malware processes that were not identified by said other remote computers having that particular combination of security products.

According to a sixth aspect of the present invention, there is provided a method of classifying a computer object as malware or as safe, wherein said computer object is a descendant or otherwise related object of a first computer object, the method comprising:

classifying a first computer object as malware or as safe;

identifying in a key relating to said first computer object a component that uniquely identifies the first computer object and that is inherited or otherwise present in the key of a descendant or other related computer object of the first computer object; and,

classifying said computer object as malware or as safe as the case may be on the basis of the unique identifier component being present in the key of said computer object.

This aspect uses the concept of ancestry to enable objects to be marked as malware. For example, any particular process may spawn child processes which are therefore related. The key relating to the first object may be inspected to identify a component that uniquely identifies the first object and that is inherited or otherwise present in the key of a descendant or other related object of the first object.

The method may comprise monitoring activities of said first computer object and reclassifying the first computer object as malware in the case that it was initially classified as safe and subsequently determined to be malware, the method further comprising automatically classifying as malware any computer object that has a key in which said unique identifier component is present.

According to a seventh aspect of the present invention, there is provided apparatus for classifying a computer object as malware or as safe, wherein said computer object is a descendant or otherwise related object of a first computer object, the apparatus comprising:

a computer constructed and arranged to classify a first computer object as malware or as safe;

the computer being constructed and arranged to identify in a key relating to said first computer object a component that uniquely identifies the first computer object and that is inherited or otherwise present in the key of a descendant or other related computer object of the first computer object; and,

the computer being constructed and arranged to classify said computer object as malware or as safe as the case may be on the basis of the unique identifier component being present in the key of said computer object.

According to an eighth aspect of the present invention, there is provided a method of installing software on a computer, the method comprising:

on initiation of installation of software on a computer, providing a computer-generated prompt on the computer to a user to ascertain whether the user authorises the installation; and,

ceasing the installation if a user authorisation is not received, else:

receiving at the computer the user's authorisation to proceed with the installation;

proceeding with the installation;

obtaining data about computer objects that are created or used during the installation;

storing said data at the local computer.

This provides for security when a user is installing new software and is not for example connected to a base computer having a community database of the type mentioned above. In that case, the method, which may be implemented in agent software running on the local computer, allows the user to permit the installation to proceed whilst at the same time gathering data about the objects (such as processes, new files, etc.) that are created during the installation.

Preferably, the locally stored data is referred to during the installation to ensure that all objects created or used during the installation are part of the installation process, and, if it is found that objects created or used during the installation are not part of the installation process, either or both of: (i) ceasing the installation and (ii) providing a computer-generated prompt on the computer to the user accordingly. This allows the method to ensure that only those objects that are

required for the installation are permitted to be created or used and thus avoids unwittingly allowing malware to install (given that malware often creates objects that are not expected as part of a normal installation of new software).

In a preferred embodiment, the method comprises connecting the computer to a community database that is connectable to a plurality of computers, and uploading the stored data to the community database for comparison with similar data provided by other computers.

The method may comprise downloading data about trusted installers to the computer, said data about trusted installers being referred to during the installation such that any objects relating to or created by the trusted installer are automatically authorised to proceed. This facilitates installation of software that is known a priori to be trustworthy.

Said data about trusted installers may be referred to only for a predetermined time period following receipt at the computer of the user's authorisation to proceed with the installation.

The present invention also includes computer programs comprising program instructions for causing a computer to perform any of the methods described above.

Although the embodiments of the invention described with reference to the drawings comprise computer processes performed in computer apparatus and computer apparatus itself, the invention also extends to computer programs, particularly computer programs on or in a carrier, adapted for putting the invention into practice. The program may be in the form of source code, object code, a code intermediate source and object code such as in partially compiled form, or in any other form suitable for use in the implementation of the processes according to the invention. The carrier be any entity or device capable of carrying the program. For example, the carrier may comprise a storage medium, such as a ROM, for example a CD ROM or a semiconductor ROM, or a magnetic recording medium, for example a floppy disk or hard disk. Further, the carrier may be a transmissible carrier such as an electrical or optical signal which may be conveyed via electrical or optical cable or by radio or other means.

Embodiments of the present invention will now be described by way of example with reference to the accompanying drawings, in which:

FIG. **1** shows schematically apparatus in which an embodiment of the present invention may be implemented;

FIG. **2** is a flowchart showing schematically the operation of an example of a method according to an embodiment of the present invention;

FIG. **3** is a flowchart showing schematically the operation of another example of a method according to an embodiment of the present invention; and,

FIG. **4** is a flowchart showing schematically an information obtaining stage.

Referring to FIG. **1**, a computer network is generally shown as being based around a distributed network such as the Internet **1**. The present invention may however be implemented across or use other types of network, such as a LAN. Plural local or "remote" computers **2** are connected via the Internet **1** to a "central" or "base" computer **3**. The computers **2** may each be variously a personal computer, a server of any type, a PDA, mobile phone, an interactive television, or any other device capable of loading and operating computer objects. An object in this sense may be a computer file, part of a file or a sub-program, macro, web page or any other piece of code to be operated by or on the computer, or any other event whether executed, emulated, simulated or interpreted. An object **4** is shown schematically in the figure and may for example be downloaded to a remote computer **2** via the Internet **1** as shown by lines **5** or applied directly as shown by line **6**.

In one preferred embodiment, the base computer **3** holds a database **7** with which the remote computers **2** can interact when the remote computers **2** run an object **4** to determine whether the object **4** is safe or unsafe. The community database **7** is populated, over time, with information relating to each object run on all of the connected remote computers **2**. As will be discussed further below, data representative of each object **4** preferably takes the form of a so-called signature or key relating to the object and its effects. As will also be discussed further below, the database **7** may further include a mask for the object **4** that sets out the parameters of the object's performance and operation.

Referring now to FIG. **2**, at the start point **21**, a computer object **4** such as a process is run at a remote computer **2**. At step **22**, by operation of local "agent" software running on the remote computer **2**, the operation of the process is hooked so that the agent software can search a local database stored at the remote computer **2** to search for a signature or key representing that particular process, its related objects and/or the event. If the local signature is present, it will indicate either that the process is considered to be safe or will indicate that that process is considered unsafe. An unsafe process might be one that has been found to be malware or to have unforeseen or known unsafe or malevolent results arising from its running. If the signature indicates that the process is safe, then that process or event is allowed by the local agent software on the remote computer **2** to run at step **23**. If the signature indicates that the process is not safe, then the process or event is stopped at step **24**.

It will be understood that there may be more than two states than "safe" or "not-safe" and choices may be given to the user. For example, if an object is considered locally to be not safe, the user may be presented with an option to allow the related process to run nevertheless. It is also possible for different states to be presented to each remote computer **2**. The state can be varied by the central system to take account of the location, status or ownership of the remote computer or timeframe.

If the object is unknown locally, then details of the object are passed over the Internet **1** or other network to the base computer **3** for storing in the community database **7** and preferably for further analysis at the base computer **3**. In that case, the community database **7** is then searched at step **25** for a signature for that object that has already been stored in the community database **7**. The community database **7** is supplied with signatures representative of objects, such as programs or processes, run by each monitored remote computer **2**. In a typical implementation in the field, there may be several thousands or even millions of remote computers **2** connected or connectable to the base computer **3** and so any objects that are newly released upon the Internet **1** or that otherwise are found on any of these remote computers **2** will soon be found and signatures created and sent to the base computer **3** by the respective remote computers **2**.

When the community database **7** is searched for the signature of the object that was not previously known at the remote computer **2** concerned, then if the signature is found and indicates that that object is safe, then a copy of the signature or at least a message that the object is safe is sent to the local database of the remote computer **2** concerned at step **26** to populate the local database. In this way, the remote computer **2** has this information immediately to hand the next time the

object **4** is encountered. A separate message is also passed back to the remote computer **2** to allow the object to run in the current instance.

If the signature is found in the community database **7** and this indicates for some reason that the object is unsafe, then again the signature is copied back to the local database and marked "unsafe" at step **27**, and/or a message is sent to the remote computer **2** so that running of the object is stopped (or it is not allowed to run) and/or the user given an informed choice whether to run it or not.

If after the entire community database **7** has been searched the object is still unknown, then it is assumed that this is an entirely new object which has never been seen before in the field. A signature is therefore created representative of the object at step **28**, or a signature sent by the remote computer **2** is used for this purpose, and this signature is initially marked as bad or unsafe community database **7** at step **29**. The signature is copied to the local database of the remote computer **2** that first ran the object at step **30**. A message may then be passed to the remote computer **2** to instruct the remote computer **2** not to run the object or alternatively the user may be given informed consent as to whether to allow the object to run or not. In addition, a copy of the object itself may be requested at step **31** by the community database **7** from the remote computer **2**.

If the user at the remote computer **2** chooses to run a process that is considered unsafe because it is too new, then that process may be monitored by the remote computer **2** and/or community database **7** and, if no ill effect occurs or is exhibited after a period of time of n days for example, it may then be considered to be safe. Alternatively, the community database **7** may keep a log of each instance of the process which is found by the many remote computers **2** forming part of the network and after a particular number of instances have been recorded, possibly with another particular number of instances or the process being allowed to run and running safely, the signature in the community database **7** may then be marked as safe rather than unsafe. Many other variations of monitoring safety may be done within this concept.

The details of an object **4** that are passed to the base computer **3** are preferably in the form of a signature or "key" that uniquely identifies the object **4**. This is mainly to keep down the data storage and transmission requirements. This key may be formed by a hashing function operating on the object at the remote computer **2**.

The key in the preferred embodiment is specially arranged to have at least three severable components, a first of said components representing executable instructions contained within or constituted by the object, a second of said components representing data about said object, and a third of said components representing the physical size of the object. The data about the object in the second component may be any or all of the other forms of identity such as the file's name, its physical and folder location on disk, its original file name, its creation and modification dates, vendor, product and version and any other information stored within the object, its file header or header held by the remote computer **2** about it; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers. In general, the information provided in the key may include at least one of these elements or any two or more of these elements in any combination.

In one preferred embodiment, a check sum is created for all executable files, such as (but not limited to) .exe and .dll files, which are of the type PE (Portable Executable file as defined by Microsoft). Three types of checksums are generated depending on the nature of the file:

Type 1: five different sections of the file are check summed. These include the import table, a section at the beginning and a section at the end of the code section, and a section at the beginning and a section at the end of the entire file. This type applies to the vast majority of files that are analysed;

Type 2: for old DOS or 16 bit executable files, the entire file is check summed;

Type 3: for files over a certain predefined size, the file is sampled into chunks which are then check summed. For files less than a certain predefined size, the whole file is check summed.

For the check summing process, in principle any technique is possible. The MD5 (Message-Digest algorithm 5) is a widely-used cryptographic hash function that may be used for this purpose.

This allows a core checksum to be generated by viewing only the executable elements of the checksum and making a comparison between two executables that share common executable code.

For the type 1 checksum mentioned above, three signature processes may be used. The first defines the entire file and will change with almost any change to the file's content. The second attempts to define only the processing instructions of the process which changes much less. The third utilises the file's size, which massively reduces the potential of collisions for objects of differing sizes. By tracking the occurrences of all signatures individually appearing with different counterparts, it is possible to identify processes that have been changed or have been created from a common point but that have been edited to perform new, possibly malevolent functionality.

This "meta data" enables current and newly devised heuristics to be run on the data in the community database **7**.

The data stored in the community database **7** provides an extensive corollary of an object's creation, configuration, execution, behaviour, identities and relationships to other objects that either act upon it or are acted upon by it.

The preferred central heuristics use five distinct processes to establish if an object is safe, unsafe or suspicious.

The first of the said processes utilises the singularity or plurality of names, locations, vendor, product and version information captured and correlated from all of the remote computers **2** that have seen the object. By considering the plurality of this information for a single object, a score can be determined which can be used as a measure of the authenticity and/or credibility of the object. Most safe objects tend not to use a large plurality of identifying information or locations. Rules can be established to consider this information in respect of the type of object and its location. For example, temporary files often utilise a plurality of system generated file names which may differ on each remote computer for the same object. Where an object has little plurality, then it provides a reference point to consider its behaviour in comparison to the known behaviours of other objects that have previously used that identifying information. For example, a new object that purports to be a version of notepad.exe can have its behaviour compared with the behaviour of one or more other objects that are also known as notepad.exe. This comparison may be against a single other object or multiple other objects that use the same or even similar identifying information. In this way, new patterns of behaviour can be identified for the new object. Also it allows the preferred embodiment to police an object's behaviour over time to identify new behavioural patterns that may cause an object that was previously considered safe to have its status reconsidered. Alternatively, the score based on identities may be considered along with scores

for other objects or scores created by other processes on this object to be considered in combinations.

The second of the said processes utilises an object's relationship to other objects that act upon it or upon which it acts. For example, analysis can be made of which object created this object, which objects this object created, which objects created a registry key to run or configure this object, which objects were configured by or had registry keys created by this object, etc. In this regard an object is considered to have a relationship based on the event performed by it upon another object, or upon it by another object. This simple 1-to-1 relationship chain provides a complex series of correlation points, allowing ancestral relationships to be considered for any object and its predecessors by event or its issue (i.e. child and sub-child processes) by event. This allows a score to be developed that describes its relationships and associations with known, unknown, known safe or known bad objects or a combination thereof. Objects that have specific relationships, volumes of relationships or mixes of relationships to one type or another may be judged safe or unsafe accordingly. Alternatively, the relationship-based score may be considered along with other scores to arrive at a determination of safe or unsafe. This data can also be used to deduce a number of factors about objects related directly or via other objects and their behaviours. For example it is possible to deduce how one object's behaviour can be influenced or changed by its association or linkage to another. Consider for example notepad.exe as supplied by Microsoft with the Windows series of operating systems. It has a limited range of functionality and would not be expected therefore to perform a wide variety of events, such as transmitting data to another computer or running other programs etc. However, the behaviour of notepad.exe could be modified by injecting new code into it, such as via dynamic link library injection (DLL injection). In this case notepad.exe would now have new capabilities derived by the code injection or linkage to another object. Using the data that defines the relationships between objects it is possible to deduce that the new behaviours of a program can be attributed to the association with another object. If that new behaviour is malevolent, then it is possible to mark either or all processes as unsafe as appropriate.

The combination of behaviours captured provide a basis to determine if the object is safe or unsafe. Malware typically exhibit certain behaviour and characteristics. For example, malware frequently has a need to self-persist. This manifests itself in the need to automatically restart on system restarts or upon certain events. Creating objects in specific locations to auto restart or trigger execution is a typical characteristic of malware. Replacing core objects of the Windows system environment are another example of typical characteristics of malware. By providing a pattern of behaviour, the determination of objects to be unsafe or safe can be automated. The centralisation of the community data in the community database **7** provides the ability to rapidly assimilate object behaviours, allowing for the rapid identification and determination of malware. Objects may also perform events upon themselves which can be considered in deriving a score.

The third said process involves time and volumes. Relevant data includes when the object was first seen, when it was last seen, how many times it has been seen, how many times it has been seen in a given interval of time, and the increase or decrease of acceleration in it being seen. This information is highly relevant in determining the prevalence of an object in the community of remote computers. A score is developed based on these metrics which can be used to determine if an object is safe, unsafe or too prevalent to allow it to execute or propagate without very thorough examination. In this case, the object can be temporarily held or blocked from executing pending further information about its behaviour or relationships. This score may also be used in combination with scores from other processes. Time is also highly relevant in combination with other information, including but not limited to behaviours and identities. For example in the case of polymorphic or randomly named objects, time is a powerful qualifier. (A polymorphic virus changes its encryption algorithm and the corresponding encryption keys each time it replicates from one computer to another and so can be difficult to detect by conventional measures.) A program that creates other programs can often be considered normal or abnormal based on its activity over time.

The fourth said process considers the behaviour of an object. This allows a score to be developed based on the types of events performed by an object or events performed on it by itself or other objects. The centralised system of the community database **7** allows for an unlimited number of event types and can consider the object performing the event or the object having the event performed upon it, or both. Some event types also relate to external information other than objects, for example a program performing an event to connect with an Internet Chat Relay site, or a program modifying a non-executable file such as the Windows hosts file. The behavioural events of an object, be they as "actor" (i.e. the object doing something to another object) or as "victim" (i.e. the object has something done to it by another object) of any event can be considered in many ways, such as in combination, in sequence, in volume, in presence or absence, or in any combination thereof. The behavioural events in the preferred embodiment may have been provided by a remote computer **2** or from other external sources. The process can consider these in isolation or in combination. Furthermore it is a feature of the preferred embodiment that the behavioural events can be considered in combination with the status of other objects upon which the object acts or that act upon the object. For example, creating a program may have a different score if the program being created is safe, unsafe, new, unknown or suspicious. Similarly, a program that is created by a known bad program will likely have a different score attributed to its creation event depending on the status of the object creating it.

The fifth process considers the behaviour of a web page or script. In this respect, the web page and url combination is assigned a unique identity which allows its behaviour to be tracked as if it were an object like any other executable file. In this example, the web page may perform events that would normally be seen as events performed by the web browser (e.g. IExplore.exe or Firefox.exe). The preferred embodiment substitutes the identifying details and signatures of the web browser for the "pseudo" object identity associated with the web page being displayed or executing within the browser. In this respect, the status of the web page and/or web site to which it relates may be determined as safe, unsafe, unknown or suspicious in the same way as any other object. The web page's "pseudo" object identity also allows the preferred embodiment to block, interject or limit the functionality of that web page or web site to prevent some or all of its potentially unsafe behaviour or to provide the remote user with qualifying information to guide them about the safety of the web site, web page or their content.

Amongst other types, the types of meta data captured might be:

"Events": these define the actions or behaviours of an object acting upon another object or some other entity. The event has three principal components: the key of the object performing the act (the "Actor"), the act being performed (the

"Event Type"), and the key of the object or identity of an other entity upon which the act is being performed (the "Victim"). While simple, this structure allows a limitless series of behaviours and relationships to be defined. Examples of the three components of an event might be:

Actor	Event Type	Victim
Object 1	Creates Program	Object 2
Object 1	Sends data	IP Address 3
Object 1	Deletes Program	Object 4
Object 1	Executes	Object 2
Object 2	Creates registry key	Object 4

"Identities": these define the attributes of an object. They include items such as the file's name, its physical location on the disk or in memory, its logical location on the disk within the file system (its path), the file's header details which include when the file was created, when it was last accessed, when it was last modified, the information stored as the vendor, the product it is part of and the version number of the file and it contents, its original file name, and its file size.

"Genesisactor"—the key of an object that is not the direct Actor of an event but which is the ultimate parent of the event being performed. For example in the case of a software installation, this would be the key of the object that the user or system first executed and that initiated the software installation process, e.g. Setup.exe.

"Ancillary data": many events may require ancillary data, for example an event such as that used to record the creation of a registry run key. In this situation the "event" would identify the Actor object creating the registry run key, the event type itself (e.g. "regrunkey"), and the Victim or subject of the registry run key. The ancillary data in this case would define the run key entry itself; the Hive, Key name and Value.

"Event Checksums": because the event data can be quite large extending to several hundred bytes of information for a single event, its identities for the Actor and Victim and any ancillary data, the system allows for this data itself to be summarised by the Event Checksums. Two event checksums are used utilising a variety of algorithms, such as CRC and Adler. The checksums are of the core data for an event. This allows the remote computer **2** to send the checksums of the data to the central computer **3** which may already have the data relating to those checksums stored. In this case, it does not require further information from the remote computer **2**. Only if the central computer **3** has never received the checksums will it request the associated data from the remote computer **2**. This affords a considerable improvement in performance for both the remote and central computers **2**, **3** allowing much more effective scaling.

Thus, the meta data derived from the remote computers **2** can be used at the community database **7** to define the behaviour of a process across the community. As mentioned, the data may include at least one of the elements mentioned above (file size, location, etc.) or two or three or four or five or six or all seven (or more elements not specifically mentioned here). This may be used accordingly to model, test and create new automated rules for use in the community database **7** and as rules that may be added to those held and used in the local database of the remote computers **2** to identify and determine the response of the remote computers **2** to new or unknown processes and process activity.

Moreover, it is possible to monitor a process along with any optional sub-processes as an homogenous entity and then compare the activities of the top level process throughout the

community and deduce that certain, potentially malevolent practices only occur when one or more specific sub-processes are also loaded. This allows effective monitoring (without unnecessary blocking) of programs, such as Internet Explorer or other browsers, whose functionality may be easily altered by downloadable optional code that users acquire from the Internet, which is of course the principal source of malevolent code today.

The potentially high volume of active users gives a high probability of at least one of them being infected by new malware. The speed of propagation can be detected and recorded so that the propagation of malware can be detected and malware designated as bad on the basis of the speed of propagation, optionally in combination with the other factors discussed above, such as file size, location and name. The simple volume of infection can also be used as a trigger. In a further embodiment, difference of naming of an otherwise identical piece of code combined with acceleration of first attempts to execute the code within the community allows pattern matching that will show up an otherwise identically signatured piece of code as bad.

This feature allows the statement in some embodiments that "nothing will propagate in our community faster than X without being locked down", so that if any process or event propagates more quickly over a given duration, it is marked as bad. This is for reasons of safety given that if for example an object is propagating quickly enough, then it might infect computers before it can be analysed to determine whether or not it is malware.

This process can be automated by the identification of the vector of propagation in the community (i.e. the source of type of propagation), from timestamp data held in the community database and the marking of a piece of code that has these attributes as bad. By comparison, it is believed that all other anti-malware providers rely on a simplistic known bad model and therefore are reliant primarily on malware infection actually occurring on terminals and being reported.

Thus, the community database **7** can be used to make early diagnosis, or simply to take precautionary measures, and thus stop potentially fast propagating worms and other malware very, very early in their life cycle. Given that it is possible to create a worm that can infect every computer connected to the Internet within a matter of a few minutes, this feature is highly desirable.

Even faster determination may be made by combining data defining the speed of propagation of a new piece of software with metadata collected by the agent software from the remote computers **2** and fed to the community database **7**. This includes monitoring processes that attempt to conceal themselves from the user by randomly changing name and/or location on the remote computers **2**. It also includes a process's attempt to create an identical copy (i.e. with identical code contents) on the computer but with a different name. This is a classic attribute of a worm.

The signature of an object may comprise or be associated with a mask which can be built up with use of that object and which indicates the particular types of behaviour to be expected from the object. If an object is allowed to run on a remote computer **2**, even if the initial signature search **22** indicates that the object is safe, then operation of that object may be monitored within the parameters of the mask. The mask might indicate for example, the expected behaviour of the object; any external requests or Internet connections that that object might legitimately have to make or call upon the remote computer **2** to make, including details of any ports or interfaces that might be required to be opened to allow such communication; any databases, either local or over a local

area network or wide area network or Internet, that may be expected to be interrogated by that object; and so on. Thus, the mask can give an overall picture of the expected "normal" behaviour of that object.

In practice, therefore, in one embodiment the behaviour of the object is continually monitored at the remote computer(s) **2** and information relating to that object continually sent to and from the community database **7** to determine whether the object is running within its expected mask. Any behaviour that extends beyond the mask is identified and can be used to continually assess whether the object continues to be safe or not. Thus, if for example the object, on a regular basis (say monthly or yearly) opens a new port to update itself or to obtain regular data, then this information is flagged. If it is found that the object has done this on other remote computers and has had no ill effects, or this behaviour is known from other objects and known to be safe, then this behaviour might be considered as safe behaviour and the mask is then modified to allow for this. If it has been found previously that this new behaviour in fact causes unsafe or malevolent results, then the object can then be marked as unsafe even if previously it was considered safe. Similarly, if the object attempts to connect to a known unsafe website, database or to take action that is known as generally being action only taken by unsafe programs, then again the object may be considered to be unsafe.

This is shown schematically in FIG. **3**. FIG. **3** also shows the concept that any object can be pre-authorised by, for example a trusted partner, such as a major software company, a verification authority, a Government department, and so on. Pre-authorisation enables a supplier of a new object, which has not been released before, to get pre-authorisation for that object, and optionally includes the provision by that supplier of a mask detailing the expected and allowable behaviour of that object.

Referring to FIG. **3** for example, when a process is run, the local and/or community databases are searched as before at step **31**. If the process is not a pre-authorised one, then the steps of FIG. **2** may be taken and the process might be allowed to run or not at step **32**. If the process is pre-authorised, as determined at step **33**, then it is immediately allowed to run, step **34**. This may terminate the operation of the method. However, in a preferred variation, the process is then monitored whilst running, and is monitored each time it is run in the future in a monitoring state step **35** to determine whether its behaviour falls within its pre-authorised mask **36**. If the behaviour falls within the pre-authorised behaviour, then the process is allowed to continue to run. If the behaviour extends beyond the allowed mask, such as by trying to instigate further processes or connections that have not been pre-authorised, then this behaviour is flagged at an alert step **37**. Various actions could be taken at this stage. The process might simply not be allowed to run. Alternatively, the trusted authority that initially enabled pre-authorisation might be contacted, who may be able to confirm that this behaviour is acceptable or not. If it is acceptable, then the mask could be modified accordingly. If not acceptable, then the process might be marked as unsafe. Many other actions may be taken upon the noting of such an alert state.

If the process has been found not to be pre-authorised at step **33** but is nevertheless allowed to run, then the process is monitored at step **38** in order to generate a mask **39** representative of the normal behaviour of that process. Data representative of this mask might be sent to the community database **7** for scanning when other computers run that process. By continually monitoring a process each time it is run or during running of the process, any behaviour that differs from previous behaviour of the process can be noted and the mask can be modified, or this behaviour might be used to determine that a process that was once considered safe should now be considered to be unsafe.

In another embodiment, a computer **2** may have agent software installed that periodically or on-demand provides information to the community database **7** that is representative of all or selected ones of the software products loaded on or available the computer **2**. In particular, this may be information on one or more of: all the locally-loaded security products (such as anti-malware systems including anti-virus software, anti-spyware, anti-adware and so on), firewall products, specific settings and details of which signature files are currently loaded, version details for the operating system and other software, and also information such as which files are operating and the particular version and software settings at any time. (It will be understood from the following that auditing and testing for a match for more of these criteria increases the likelihood of computers being very similarly arranged and thus reduces the rate of false negatives and positives during the match search.)

The information relating to these software products, etc. may be provided individually to the community database. Preferably however, again for reasons of data quantity for storage and transmission, the information is provided as a signature or key representative of the information (e.g. by a hashing or compression function at the computer **2**). FIG. **4** shows schematically how the details of all local security products, versions, signature files, firewall settings, etc. **40** are used to create a key **41**. The key is transmitted to the community database **7**. Since the community database **7** is provided with such information from many, possibly millions, of users' computers **2**, it is likely to hold corresponding information for other computers **2** that have the same or a similar configuration of security products, etc. Thus, the community database **7** can be searched at step **42** for other computers **2** having the same or a similar combination of security products including the same setting, signature files loaded and so on.

The community database **7** in this embodiment is also provided by the agent software with details of processes run by every computer **2** and thus knows whether or not a process has been detected by each computer **2**.

In this way, the community database **7** can be used to obtain information as to whether for example a particular, specific combination of operating system and various security products, including settings and signature files existing at a point in time, renders a particular computer **2** having those products and settings susceptible or vulnerable to any particular malware object.

In a simple example, if for example the database knows that a computer in the past has version A of anti-virus product B with downloaded signature update C, and also has a firewall D with particular settings E, and perhaps anti-spyware software F with signature updates G, but that a particular malware process P was not detected by this combination of programs at that point in time, then this information can be provided to a computer **2** that is known as having that combination of security programs/settings, and can be used to indicate that that computer **2** is vulnerable in the short term to attack by that particular malware process. This information can be presented to the user either by displaying a window **43** on the screen display of the computer **2** or by directing the user to a particular website which explains the position in more detail. The user might be informed for example that their particular combination of security products, etc., exposes their computer to a risk of being infected by the Sobig virus as that virus is not detectable by their computer. The user might be offered

specific advice (e.g. to update a particular anti-virus program with a particular signature file) or software to download and install to remove the risk.

Thus, the community database **7**, when provided with information relating to all the security products, etc. on a particular computer at a particular time, is searched for events for processes marked as "bad" that occurred on computers with that particular mix of security products and that were not locally detected. This information can then be fed back to the user of the particular computer, for example directly or by directing the user to a website. This information can be provided virtually in real-time, allowing a new user or a user of a new computer to be able to increase the computer's effective security very quickly.

The preferred method also tracks which objects are related to each other and uses the concept of ancestry to enable objects to be marked as malware. For example, any particular process may spawn child processes which are therefore related. The key relating to the first object may be inspected to identify a component that uniquely identifies the first object and that is inherited or otherwise present in the key of a descendant or other related object of the first object. This component is referred to herein as a "gene". This general technique may be used in a number of ways:

a) A known and trusted parent process is afforded the ability to create child processes which may be automatically marked as safe to run on the local computer. It is also possible that this "inherited" property may be passed down to grand children processes and so on. This safe status is passed to the parent's child processes and possibly, through them, further child processes (referred to here as "issue"), such signatures for the issue can all automatically be recorded in the local database as good. This allows the issue processes to be quickly marked as good, even if a connection to the community database **7** is not available.

b) By monitoring activity of the parent process, if it is later found that the parent process is malware, then all of the issue processes can all automatically be recorded in the local database as bad.

c) Similarly, by monitoring activity of the issue processes, if it is later found that one of the issue processes is malware, then one or more of the parent process and all of the other issue processes (i.e. all of the related processes in this context) can all automatically be recorded in the local database as bad.

d) Parental creation of a signature for a child or children including the ability for these to be automatically marked as either good or bad depending on the parent's behaviour and determination. Note that in some embodiments the product can "watch" or monitor the birth of a child process and automatically create the signature upon arrival. This provides the ability to monitor the creation of a bad program by another bad program. It is possible therefore to monitor the ancestry of a program so if for example the grandfather creates a program (the father) and this in turn creates a bad program (the son), it is possible automatically to determine the father as a bad program.

e) A feature may be included that allows for automatic forfeiture of a child's inherited ability to trigger the automatic creation of signatures on any further births because the child, as parent, has already produced bad offspring. Preferably, a rule is that if a file has one bad offspring then the inherited ability can be automatically removed.

f) An ability to watch clones or identical twins of objects (e.g. the same process running on other systems in the community) to compare the pattern of their issue and to make decisions as to whether or not to treat any particular process as malware.

One or more of these features a) to f) can be used to provide a solution to the problem of producing a security product that can be used effectively without 100% reliance on being permanently connected to the Internet, which is often impractical. Examples of this are Windows Update and other processes used more and more by vendors who wish to be able to roll out product updates automatically across the worldwide web.

Possible benefits of these types of features above conventional software are as follows. Antivirus software tends to have a cache of known bad signatures. The problem is keeping this up to date. Take the simple example of someone buying a new computer. The computer comes with an antivirus product preloaded with a signature cache. Between the time when the PC was built, shipped to the store and bought by the user several days or weeks will have passed. The user starts the PC and is exposed to any new virus or malware which was created after the PC was built. Full protection requires the user to connect to the internet and download updates. This cannot be guaranteed to occur ahead of other activities by the user on the internet (almost physically impossible to guarantee). With a local cache of known good processes, as in the embodiments of the present invention, it is possible to ship the computer/terminal preloaded with a pre-generated cache of signatures for all of the good (clean) software preloaded by the computer manufacturer. In this case the user can connect to the internet knowing that any new or updated programs will be immediately detected and verified. Also any auto-updating software can function forcing signatures to be automatically built for its children and more remote off-spring (i.e. grandchildren, great-grandchildren, etc).

Reference is now made to "Defeating Polymorphism: Beyond Emulation" by Adrian Stepan of Microsoft Corporation published in "Virus Bulletin Conference October 2005" and also to U.S. 60/789,156 filed on 5 Apr. 2006, the entire content of which are hereby incorporated by reference. In that paper and patent application, there are disclosed methods of decrypting files to allow the files to be analysed to determine whether or not the file actually is malware. In U.S. 60/789,156 in particular, there is disclosed a method of unpacking/decrypting an executable computer file using a host computer, the method comprising: partitioning the executable computer file into plural basic blocks of code; translating at least some of the basic blocks of code into translated basic blocks of code that can be executed by the host computer; linking at least some of the translated basic blocks of code in memory of the host computer; and, executing at least some of the translated basic blocks of code on the host computer so as to enable the executable computer file to be unpacked or decrypted, whereupon the unpacked or decrypted executable computer file can be analyzed to determine whether the executable computer file is or should be classed as malware. There is also disclosed in U.S. 60/789,156 a method of unpacking/decrypting an executable computer file, the method comprising: partitioning the executable computer file into plural basic blocks of code; creating at least a read page of cache memory for at least some of the basic blocks, the read page of cache memory storing a read cached real address corresponding to a read cached virtual memory address for the respective basic block, and creating at least a write page of cache memory for at least some of the basic blocks, the write page of cache memory storing a write cached real address corresponding to a write cached virtual memory address for the respective basic block; and, emulating the executable file by executing at least some of the basic blocks of code so as to enable the executable computer file to be unpacked or decrypted, whereupon the

unpacked or decrypted executable computer file can be analyzed to determine whether the executable computer file is or should be classed as malware; wherein during the execution of a basic block, at least one of the read page and the write page of cache memory is checked for a cached real address corresponding to the virtual address that is being accessed for said basic block.

The techniques disclosed in these documents can be used in the present context when it is desired to analyse a file in detail. More generally however, the techniques disclosed in these papers, and particularly the enhanced techniques disclosed in U.S. 60/789,156, can be used to provide information about the activity of a file when it is run on a computer because the techniques disclosed in these papers emulate the running of the file and therefore allow the file's activity to be interpreted.

A further situation arises when users wish to install software while offline. In a preferred embodiment, when a user attempts to install new software while offline, agent software running on the user's computer **2** prompts the user for authorisation to allow the installation process to proceed such that the execution of the installation can itself be "authorised" by the user. This authorisation by the user is treated as a "Genesis" event and will be so termed hereinafter. However there are some processes commonly used in installation of software that communicate with other existing programs on the installation machine, e.g. Microsoft's MSIEXEC.

The Genesis approach involves a process that generates signatures as a result of the user's authorisation on the user's computer **2**. Those signatures are stored in the local database on the user's computer **2**. In one embodiment, those locally stored signatures are referred to as necessary by the agent software during the installation process so that the installation can proceed. Alternatively, the security checks made by the agent software can be switched off during the installation process. The switching off may only be for a limited duration, such as a few minutes which should be sufficient to allow most software to be installed, the off time optionally being user-configurable.

In any event, once the installation has been completed and the user's computer **2** connected to the community database **7**, the agent software on the user's computer **2** can upload the signatures relating to the installation from the user's local database to the community database **7**. With corresponding data from other users' computers **2**, the community database **7** can then be used to make a rapid determination that the installation of this particular software is benign.

As a variant to this embodiment, when a user's computer **2** is at some point in time on-line, the agent software on the user's computer **2** may download signatures of a so-called "trusted installer" or "licensed installer". This allows the operation of a method such that a "licensed installer" and any child processes of the licensed installer are permitted to execute while a Genesis event is "current", e.g. within a period of minutes after an authorisation from the user. Preferably signatures of licensed installers are always downloaded, as and when added to the community database **7**, to a remote computer **2** while online.

There may be further refinements to this method, such as to prevent licensed installers executing if a media change has occurred during the currency of a Genesis event. However, any unknown processes, which may be malware, may still be detected and blocked. Having a small number of licensed installers facilitates download, as minimal data is required, especially compared to downloading virus signature files. As another example, super-trusted installers, such as for example "Windows Update", may be employed whereby all new processes created by the super-trusted installer are marked immediately as safe.

In the case of an installation occurring when the remote computer **2** is connected to the community database **7**, another option is that if the software that is about to be installed is not known to the community database **7**, then the system will block the installation or alert the user. For example, a message may be displayed at the user's computer **2** to the effect that "You are about to install some software. This software is not known to [the community]. Are you sure you wish to proceed?".

Embodiments of the present invention have been described with particular reference to the examples illustrated. However, it will be appreciated that variations and modifications may be made to the examples described within the scope of the present invention.

The invention claimed is:

1. A method of classifying a computer object as malware, the method comprising:

at a base computer, receiving data about a computer object from each of plural remote computers on which the object or similar objects are stored, the data including information about the behaviour of the object running on one or more remote computers;

determining in the base computer whether the data about the computer object received from the plural computers indicates that the computer object is malware;

classifying the computer object as malware when the data indicates that the computer object is malware;

when the determining does not indicate that the computer object is malware, initially classifying the computer object as not malware;

automatically generating a mask for the computer object that defines acceptable behaviour for the computer object, wherein the mask is generated in accordance with normal behaviour of the object determined from said received data;

running said object on at least one of the remote computers;

automatically monitoring operation of the object on the at least one of the remote computers;

allowing the computer object to continue to run when behaviour of the computer object is permitted by the mask;

disallowing the computer object to run when the actual monitored behaviour of the computer object extends beyond that permitted by the mask; and,

reclassifying the computer object as malware when the actual monitored behaviour extends beyond that permitted by the mask.

2. A method according to claim **1**, wherein the data about the computer object that is sent from the plural remote computers to the base computer includes one or more of: executable instructions contained within or constituted by the object; the size of the object; the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

3. A method according to claim **1**, wherein determining identifies relationships between the object and other objects.

4. A method according to claim **3**, wherein if at least one other object to which said object is related is classed as malware, then classifying said object as malware.

5. A method according to claim **3**, wherein said other objects include the object or similar objects stored on at least some of the remote computers.

6. A method according to claim **3**, wherein said other objects include other objects that are parent objects or child objects or otherwise process-related objects to said object.

7. A method according to claim **1**, wherein the data is sent in the form of key that is obtained by a hashing process carried out in respect of the objects on the respective remote computers.

8. A method according to claim **7**, wherein the key has at least one component that represents executable instructions contained within or constituted by the object.

9. A method according to claim **7**, wherein the key has at least one component that represents data about said object.

10. A method according to claim **9**, wherein said data about said object includes at least one of: the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

11. A method according to claim **7**, wherein the key has at least one component that represents the physical size of the object.

12. A method according to claim **1**, comprising if the monitored behaviour of the object running on the remote computer exhibits non-permitted behaviour that extends beyond that permitted by the mask, comparing at the base computer the non-permitted behaviour with the behaviour of other objects and if said behaviour is known to be not malicious for said other objects, allowing said behaviour for the object and modifying the mask to allow that behaviour for that object in future.

13. Apparatus for classifying a computer object as malware, the apparatus comprising:

a base computer constructed and arranged to receive data about a computer object from each of plural remote computers on which the object or similar objects are stored, the data including information about the behaviour of the object running on one or more remote computers;

the base computer being constructed and arranged to determine whether the data about the computer object received from said plural computers indicates that the computer object is malware; and,

the base computer being constructed and arranged to classify the computer object as malware when the base computer determines that the data indicates that the computer object is malware,

wherein the base computer is constructed and arranged to automatically generate a mask for an object that is initially classed not as malware, said mask defining acceptable behaviour for the object built in accordance with normal behaviour of the object determined from said received data, wherein operation of the object is automatically monitored when the object is running on at least one of the remote computers, the object being allowed to continue to run when behaviour of the object is permitted by the mask, the object not being permitted to run when the actual monitored behaviour of the object extends beyond that permitted by the mask, and the object is reclassified as malware when the actual monitored behaviour extends beyond that permitted by the mask.

14. Apparatus according to claim **13**, the data includes one or more of: executable instructions contained within or constituted by the object; the size of the object; the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

15. Apparatus according to claim **13**, wherein the base computer is constructed and arranged so that the determining identifies relationships between the object and other objects.

16. Apparatus according to claim **15**, the base computer is constructed and arranged so that if at least one other object to which said object is related is classed as malware, then said object is classified as malware.

17. Apparatus according to claim **15**, wherein the base computer is constructed and arranged so that said other objects include the object or similar objects stored on at least sonic of said remote computers.

18. Apparatus according to claim **15**, wherein the base computer is constructed and arranged so that said other objects include other objects that are parent objects or child objects or otherwise process-related objects to said object.

19. Apparatus according to claim **13**, wherein the base computer is constructed and arranged so as to be able to process data that is sent in the form of key that is obtained by a hashing process carried out in respect of the objects on said respective remote computers.

20. Apparatus according to claim **19**, wherein the key has at least one component that represents executable instructions contained within or constituted by the object.

21. Apparatus according to claim **19**, wherein the key has at least one component that represents data about said object.

22. Apparatus according to claim **21**, wherein said data about said object includes at least one of: the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

23. Apparatus according to claim **19**, wherein the key has at least one component that represents the physical size of the object.

24. A method according to claim **13**, wherein, if the monitored behaviour of the object running on the remote computer exhibits non-permitted behaviour that extends beyond that permitted by the mask, the base computer is arranged to compare the non-permitted behaviour with the behaviour of other objects and if said behaviour is known to be not malicious for said other objects the base computer is arranged to allow said behaviour for the object and to modify the mask to allow that behaviour for that object in future.

25. A method of providing data about a computer object from a remote computer to a base computer so that a comparison can be made at the base computer with similar data received from other remote computers, the method comprising:

providing from a remote computer to a base computer data about a computer object that is stored on the remote computer;

the data including information about the behaviour of the object running on one or more remote computers and including one or more of: executable instructions con-

tained within or constituted by the object; the size of the object; the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers;

the data being sent in the form of key that is obtained by a hashing process carried out in respect of the object on the remote computer,

automatically generating a mask for the computer object, the mask defining acceptable behaviour for the object built in accordance with normal behaviour of the object determined from said received data;

executing the computer object at the remote computer;

automatically monitoring operation of the computer object at the remote computer compared to the behaviour permitted by the mask;

allowing the object to continue to run when behaviour of the computer object is permitted by the mask;

disallowing the object to run when the actual monitored behaviour of the object extends beyond that permitted by the mask; and,

reclassifying the object as malware when the actual monitored behaviour extends beyond hat permitted by the mask, wherein the base computer is arranged to receive information about the behaviour of the object running on one or more remote computers, wherein the step of generating a mask for that object comprises building the mask with the base computer in accordance with normal behaviour of the object determined from said received information.

26. A method according to claim **25**, wherein the key has at least one component that represents executable instructions contained within or constituted by the object.

27. A method according to claim **25**, wherein the key has at least one component that represents data about said object.

28. A method according to claim **27**, wherein said data about said object includes at least one of: the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

29. A method according to claim **25**, wherein the key has at least one component that represents the physical size of the object.

30. A non-transitory storage medium storing a computer program comprising program instructions for causing a computer to perform the method of claim **25**.

* * * * *

Exhibit 2

U 8164950

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

October 07, 2021

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THIS OFFICE OF:

U.S. PATENT: *8,726,389*
ISSUE DATE: *May 13, 2014*

By Authority of the
Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office

R GLOVER
Certifying Officer

US008726389B2

(12) **United States Patent**
Morris et al.

(10) **Patent No.: US 8,726,389 B2**
(45) **Date of Patent: May 13, 2014**

(54) **METHODS AND APPARATUS FOR DEALING WITH MALWARE**

(75) Inventors: **Melvyn Morris**, Turnditch (GB); **Paul Stubbs**, Wyboston (GB); **Markus Hartwig**, Milton Keynes (GB); **Darren Harter**, Hucclecote (GB)

(73) Assignee: **Prevx Limited** (GB)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **13/543,866**

(22) Filed: **Jul. 8, 2012**

(65) **Prior Publication Data**

US 2012/0278895 A1 Nov. 1, 2012

Related U.S. Application Data

(63) Continuation of application No. 11/477,807, filed on Jun. 30, 2006.

(30) **Foreign Application Priority Data**

Jun. 30, 2005 (GB) 0513375.6

(51) **Int. Cl.**
G06F 11/00 (2006.01)
G06F 7/04 (2006.01)
G06F 12/14 (2006.01)
G06F 17/30 (2006.01)

(52) **U.S. Cl.**
USPC **726/24**; 726/22; 726/23; 726/25; 726/26

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,338,141 B1	1/2002	Wells	
6,772,346 B1	8/2004	Chess	
6,944,772 B2	9/2005	Dozortsev	
7,013,483 B2	3/2006	Cohen et al.	
7,093,239 B1	8/2006	van der Made	
7,603,374 B2 *	10/2009	Cameron et al.	1/1

(Continued)

FOREIGN PATENT DOCUMENTS

EP	1315066	5/2003
EP	1280040	3/2004

(Continued)

OTHER PUBLICATIONS

U.S. Appl. No. 60/786,156, filed Apr. 5, 2006, Chiriac.
English Language Abstract of JP 3-233629 published Oct. 17, 1991.
English Language Abstract and Translation of JP 6-110718 published Apr. 22, 1994.

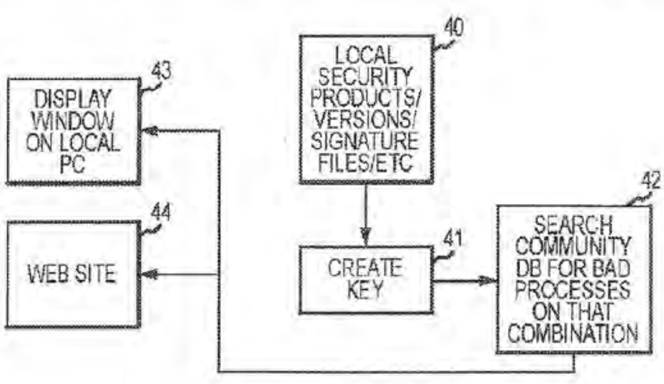
(Continued)

Primary Examiner — Cordelia Zecher
Assistant Examiner — Trang Doan
(74) *Attorney, Agent, or Firm* — Sheridan Ross P.C.

(57) **ABSTRACT**

In one aspect, a method of classifying a computer object as malware includes receiving at a base computer data about a computer object from each of plural remote computers on which the object or similar objects are stored. The data about the computer object received from the plural computers is compared in the base computer. The computer object is classified as malware on the basis of said comparison. In one embodiment, the data about the computer object includes one or more of: executable instructions contained within or constituted by the object; the size of the object; the name of the object; the logical storage location or path of the object on the respective remote computers; the vendor of the object; the software product and version associated with the object; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

30 Claims, 3 Drawing Sheets



Copy provided by USPTO from the PIRS Image Database on 10-05-2021

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,793,338	B1	9/2010	Beddoe et al.	
2001/0052014	A1	12/2001	Sheymov et al.	
2002/0087734	A1 *	7/2002	Marshall et al.	709/310
2002/0099952	A1	7/2002	Lambert et al.	
2002/0194490	A1	12/2002	Halperin et al.	
2003/0023857	A1	1/2003	Hinchlife	
2003/0084323	A1	5/2003	Gales	
2003/0101381	A1	5/2003	Mateev	
2003/0131256	A1	7/2003	Ackroyd	
2003/0135791	A1	7/2003	Natvig	
2003/0177394	A1 *	9/2003	Dozortsev	713/201
2003/0195861	A1	10/2003	McClure et al.	
2004/0006704	A1 *	1/2004	Dahlstrom et al.	713/200
2004/0039921	A1	2/2004	Chuang	
2004/0068618	A1	4/2004	Hooker	
2004/0073810	A1	4/2004	Racine	
2004/0083408	A1	4/2004	Spiegel	
2004/0153644	A1	8/2004	McCorkendale	
2004/0255165	A1	12/2004	Szor	
2005/0021994	A1 *	1/2005	Barton et al.	713/200
2005/0022014	A1	1/2005	Shipman	
2005/0027686	A1	2/2005	Shipp	
2005/0086500	A1	4/2005	Albornoz	
2005/0210035	A1 *	9/2005	Kester et al.	707/10
2006/0085857	A1 *	4/2006	Omote et al.	726/24
2007/0016953	A1	1/2007	Morris et al.	
2008/0209562	A1	8/2008	Szor	
2008/0320595	A1	12/2008	van der Made	
2009/0271867	A1	10/2009	Zhang	
2012/0278891	A1	11/2012	Morris et al.	

FOREIGN PATENT DOCUMENTS

EP	1536341	6/2005
EP	1549012	6/2005
JP	3-233629	10/1991
JP	6-110718	4/1994
JP	9-504395	4/1997
JP	2003-196112	7/2003
WO	WO 95/12162	5/1995
WO	WO 96/30829	10/1996
WO	WO 99/15966	4/1999
WO	WO 02/33525	4/2002
WO	WO 03/021402	3/2003
WO	WO 2004/097602	11/2004

OTHER PUBLICATIONS

English Language Abstract of JP 2003-196112 published Jul. 11, 2003.
Related U.S. Appl. No. 11/694,261.
Zenkin "Fighting Against the invisible Enemy—Methods for detecting an unknown virus," Computers & Security, Elsevier Science Publishers, Amsterdam, NL, vol. 20, No. 4, Jul. 31, 2001, pp. 316-321, XP004254268.
International Search Report issued in PCT/GB2006/002439 issued Jan. 15, 2007.
English Translation of Official Action for China Patent Application No. 201110036121.8, dated Sep. 24, 2012, 8 pages.
Official Action with English translation for China Patent Application No. 201110036121.8, dated Oct. 10, 2011, 14 pages.
Japanese Office Action issued in JP 2008-518975 on Sep. 13, 2011.
English Language Translation of Japanese Office Action issued in JP 2008-518975 on Sep. 13, 2011.
Official Action for U.S. Appl. No. 11/477,807, mailed Jan. 13, 2010, 8 pages.
Official Action for U.S. Appl. No. 11/477,807, mailed May 4, 2010, 13 pages.
Official Action for U.S. Appl. No. 11/477,807, mailed Sep. 28, 2010, 11 pages.
Official Action for U.S. Appl. No. 11/477,807, mailed Feb. 15, 2011, 13 pages.
Official Action for U.S. Appl. No. 11/477,807, mailed Jul. 25, 2011, 13 pages.
Notice of Allowance for U.S. Appl. No. 11/477,807, mailed Mar. 21, 2012, 9 pages.
Notice of Allowance for U.S. Appl. No. 11/477,807, mailed Dec. 13, 2012, 7 pages.
Official Action for China Patent Application No. 201110036124.1, dated Apr. 17, 2013, 7 pages.
Official Action for U.S. Appl. No. 13/543,865, mailed Apr. 15, 2013, 9 pages.
Official Action for European Patent Application No. 06755686.0 dated Jul. 31, 2013, 5 pages.
Extended European Search Report for European Patent Application No. 13167434.3 dated Aug. 1, 2013, 6 pages.
Extended European Search Report for European Patent Application No. 13167436.8 dated Aug. 30, 2013, 5 pages.
Official Action for U.S. Appl. No. 13/543,865, mailed Aug. 6, 2013, 11 pages.
Official Action with English Translation for China Patent Application No. 201110036124.1, dated Oct. 23, 2013, 8 pages.

* cited by examiner

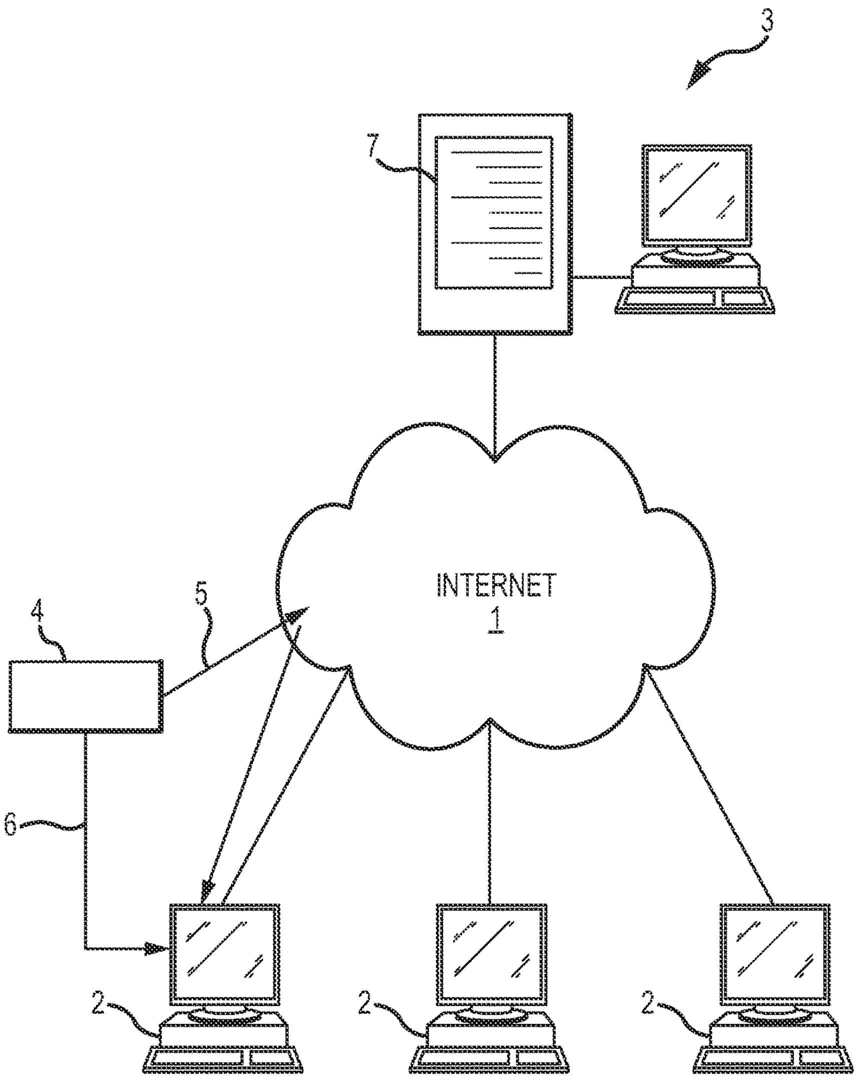


FIG.1

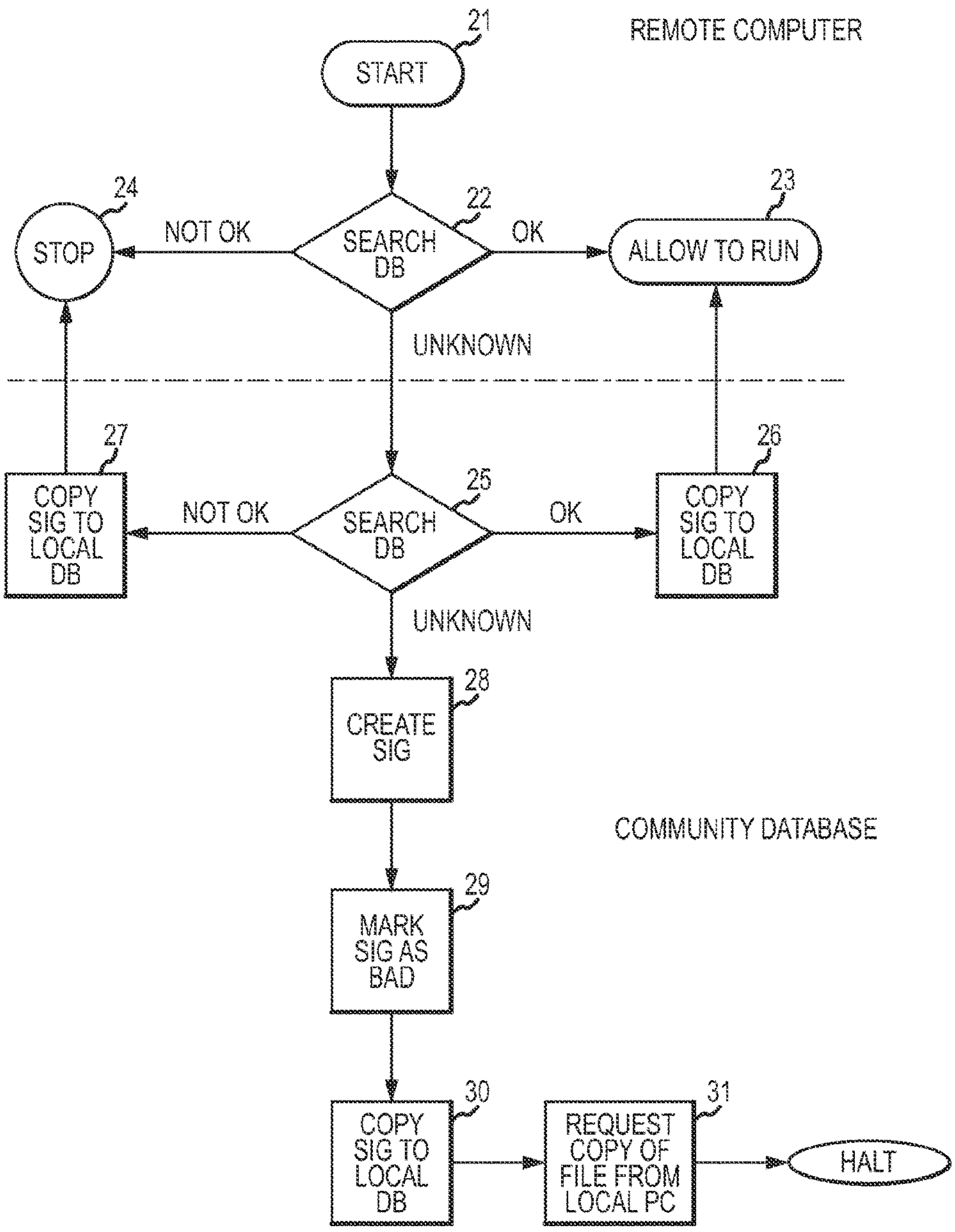

REMOTE COMPUTER
21
START
22
SEARCH DB
24
STOP
NOT OK
OK
23
ALLOW TO RUN
UNKNOWN
27
COPY SIG TO LOCAL DB
25
SEARCH DB
NOT OK
OK
26
COPY SIG TO LOCAL DB
UNKNOWN
28
CREATE SIG
COMMUNITY DATABASE
29
MARK SIG AS BAD
30
COPY SIG TO LOCAL DB
31
REQUEST COPY OF FILE FROM LOCAL PC
HALT

FIG.2

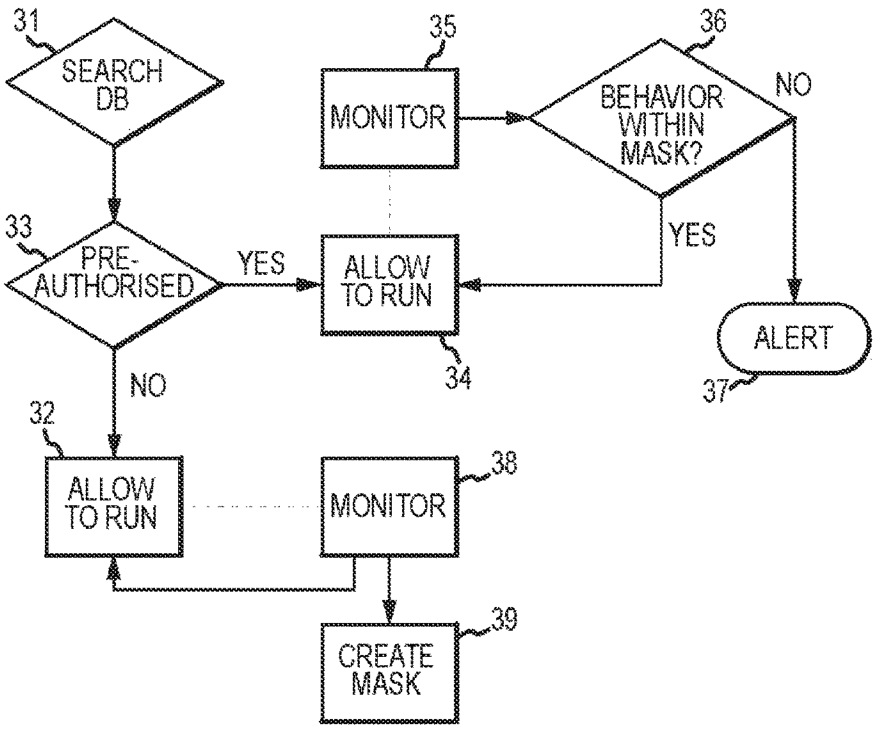

31
SEARCH DB
33
PRE-AUTHORISED
YES
NO
32
ALLOW TO RUN
35
MONITOR
ALLOW TO RUN
34
36
BEHAVIOR WITHIN MASK?
NO
YES
ALERT
37
MONITOR
38
CREATE MASK
39

FIG.3

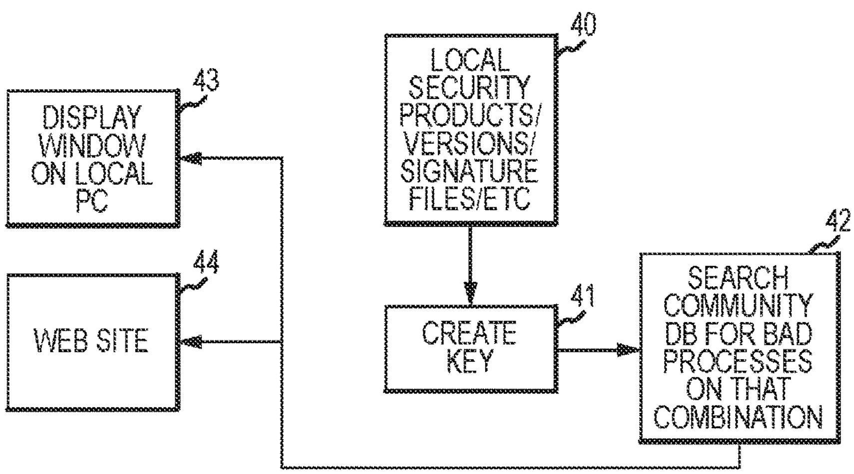

DISPLAY WINDOW ON LOCAL PC
43
WEB SITE
44
LOCAL SECURITY PRODUCTS/ VERSIONS/ SIGNATURE FILES/ETC
40
CREATE KEY
41
SEARCH COMMUNITY DB FOR BAD PROCESSES ON THAT COMBINATION
42

FIG.4

METHODS AND APPARATUS FOR DEALING WITH MALWARE

This application is a continuation of U.S. application Ser. No. 11/477,807 filed Jun. 30, 2006, which claims priority to United Kingdom Application No. 0513375.6, filed Jun. 30, 2005. The entirety of all of the above-listed Applications are incorporated herein by reference.

The present invention relates generally to methods and apparatus for dealing with malware. In one aspect, the present invention relates to a method and apparatus for classifying a computer object as malware. In another aspect, the present invention relates to a method and apparatus for determining the protection that a remote computer has from malware. In another aspect, the present invention relates to a method and apparatus for classifying a computer object as malware or as safe. In another aspect, the present invention relates to a method of installing software on a computer.

The term "malware" is used herein to refer generally to any executable computer file or, more generally "object", that is or contains malicious code, and thus includes viruses, Trojans, worms, spyware, adware, etc. and the like.

A typical anti-malware product, such as anti-virus scanning software, scans objects or the results of an algorithm applied to the object or part thereof to look for signatures in the object that are known to be indicative of the presence of a virus. Generally, the method of dealing with malware is that when new types of malware are released, for example via the Internet, these are eventually detected. Once new items of malware have been detected, then the service providers in the field generate signatures that attempt to deal with these and these signatures are then released as updates to their anti-malware programs. Heuristic methods have also been employed.

These systems work well for protecting against known malicious objects. However, since they rely on signature files being generated and/or updated, there is inevitably a delay between a new piece of malware coming into existence or being released and the signatures for combating that malware being generated or updated and supplied to users. Thus, users are at risk from new malware for a certain period of time which might be up to a week or even more. Moreover, in order to try to defeat anti-virus products, the malware writers use obfuscation techniques in order to attempt to hide the signature or signature base data of the virus code from detection. Typically, the obfuscation involves encrypting or packing the viral code.

WO-A-2004/097602 describes a system that analyses computer files received or generated by a local computer and compares these with a database of known files to determine whether a particular file is known and if so whether it has been known about long enough that it can be regarded as "safe". However, in practice, on its own this is not likely to provide for adequate protection because, for example, the active payload of a virus or Trojan may only be programmed to activate at a particular date, or upon receiving a message or instruction from a local or remote system or process, or on the occurrence of a particular event that may be many months or even years after the process has been first run or is released. Thus, just looking at the age of a file is an unsatisfactory way of determining whether it is properly safe and will remain so.

In the system of US-A-2004/0083408, a worm in a file is detected by examining connection attempts made by the specific file running on a computer.

U.S. Pat. No. 6,944,772, U.S. Pat. No. 6,772,346, EP-A-1549012 and EP-A-1280040 all disclose "community-based" anti-malware systems in which a plurality of "local" computers all connect via a network (which may be a LAN or the Internet, for example) to a central computer. On encountering a file that is not already known to them, the local computers send a request to the central computer for authorisation to run the file. If the file is recognised at the central computer, then the central computer can send permission for the local computer to run the file if the file is known to be safe or send a "deny" command if the file is known to be malicious. However, in each of these prior art proposals, if the file is not known at the central computer, then the whole file is sent to the central computer where it can be analysed to determine whether it should be regarded as safe or malware. Such analysis is typically carried out manually or "semi-manually" by subjecting the file to detailed analysis, for example by emulation or interpretation, which can still take days given the human involvement that is typically required. There is therefore still a considerable period of time before a new file is classified as safe or as malware. In the case of these prior art systems, the request for authorisation to run the file that is sent by a local computer to the central computer may comprise sending a checksum or "signature" or "key" that uniquely represents the file.

A similar community-based anti-malware system is disclosed in WO-A-02/33525. In this system, in the case that a local computer is seeking clearance to run a file that is not known by the central computer to be safe or malware, some limited audit information about the prevalence of the file on other local computers can be sent to a human system administrator associated with the local computer that is seeking permission to run the file. The human system administrator can therefore make a better informed though still "manual" decision as to whether or not the file is safe to run.

In the system of US-A-2004/0073810, a metafile containing data about an attachment or other transmitted file is sent to a central computer. The data about that file is analysed to determine a likelihood of the transmitted file being malware. A specific example given is that if the transmitted file has been transmitted via at least a certain number of servers, then it should be treated as malware.

In the systems disclosed in US-A-2005/0021994 and US-A-2004/0153644, pre-approved files, which may be certified as safe by for example the software vendor associated with the files, may be permitted always to run without further checking. In one embodiment of the system of US-A-2004/0153644, monitoring is carried out to decide that a file is malicious if an abnormally high number of requests by that file is received at a central authority from plural local computers in a time period or if an abnormally high number of requests by that file on a single computer is received from the single local computer in a time period.

In the system of US-A-2004/0006704, a comparison is made between installed versions of software on a computer with a database of software versions and their known vulnerabilities. A user of the computer can therefore be informed of specific risks and how to minimise those risks by updating existing or installing new software.

In the system of WO-A-03/021402, a central database holds a virtual image of all files stored on each of plural local computers. If a threat in one local computer is identified, other local computers with a similar configuration can be notified of the risk.

Thus, the prior art systems either rely on deep analysis of a new object in order to determine whether or not the object is malicious, which introduces delay and therefore risk to users during the period that the file is analysed and new anti-malware signatures distributed, or limited analysis of the opera-

tion of the particular object or its method of transmission to a computer is carried out to decide a likelihood of the object being malicious.

According to a first aspect of the present invention, there is provided a method of classifying a computer object as malware, the method comprising:

at a base computer, receiving data about a computer object from each of plural remote computers on which the object or similar objects are stored;

comparing in the base computer the data about the computer object received from the plural computers; and,

classifying the computer object as malware on the basis of said comparison.

Compared to the prior art that relies solely on signature matching, this aspect allows a comparison to be made between the objects and/or their effects on the different remote computers to determine whether or not a particular object should be classed as good or as malware. Sophisticated pattern analysis can be carried out. This allows a rapid determination of the nature of the object to be made, without requiring detailed analysis of the object itself as such to determine whether it malware and also avoids the need to generate new signatures to be used for signature matching as in the conventional prior art anti-virus software.

In a preferred embodiment, the data about the computer object that is sent from the plural remote computers to the base computer and that is used in the comparison includes one or more of: executable instructions contained within or constituted by the object; the size of the object; the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

Preferably, the comparing identifies relationships between the object and other objects. In an example, this can be used immediately and automatically to mark a child object as bad (or good) if the or a parent or other related object is bad (or good). Thus, if at least one other object to which said object is related is classed as malware, then the method may comprise classifying said object as malware. Said other objects include the object or similar objects stored on at least some of the remote computers. Said other objects may include other objects that are parent objects or child objects or otherwise process-related objects to said object.

In a most preferred embodiment, the data is sent in the form of key that is obtained by a hashing process carried out in respect of the objects on the respective remote computers. A major advantage of using such a key is that it keeps down the volume of data that needs to be transmitted to the base computer. Given that there may be thousands or even millions of connected remote computers and further given that each may send details about very many objects, this can be an important advantage.

The key preferably has at least one component that represents executable instructions contained within or constituted by the object. This important preferred feature allows a comparison to be made at the base computer of only the executable instructions of the object. This means for example that differently named objects that basically have the same executable instructions, which is often an indicator that the objects are malware, can nevertheless be regarded as the "same" object for this purpose. As another example, a new version of a program may be released which has minor changes compared to a previous version already known to the base computer and which in substance, at least in respect of the executable instructions, can be regarded as being the same as the previous version. In that case, the minor differences can be ignored and the objects regarded as being the same. Not only is this useful in distinguishing between malware and for example revised versions of previous software, it also keeps down the data transmission and storage requirements because the base computer can inform the remote computers that an apparently new object is for this purpose the same as a previously known object, thus avoiding having the remote computers send full details about the object or the object itself to the base computer.

The key preferably has at least one component that represents data about said object. Said data about said object may include at least one of: the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

The key preferably has at least one component that represents the physical size of the object.

Where more than one of these components are present in the key, the plural components are preferably severable.

The method may comprise initially classifying an object as not malware, generating a mask for said object that defines acceptable behaviour for the object, and comprising monitoring operation of the object on at least one of the remote computers and reclassifying the object as malware if the actual monitored behaviour extends beyond that permitted by the mask. This provides an efficient and effective way of monitoring the behaviour of an object that has been classified or regarded as good and allows the object to be reclassified quickly as malware if the behaviour of the object warrants it.

According to a second aspect of the present invention, there is provided apparatus for classifying a computer object as malware, the apparatus comprising:

a base computer constructed and arranged to receive data about a computer object from each of plural remote computers on which the object or similar objects are stored;

the base computer being constructed and arranged to compare the data about the computer object received from said plural computers; and,

the base computer being constructed and arranged to classify the computer object as malware on the basis of said comparison.

According to a third aspect of the present invention, there is provided a method of providing data about a computer object from a remote computer to a base computer so that a comparison can be made at the base computer with similar data received from other remote computers, the method comprising:

providing from a remote computer to a base computer data about a computer object that is stored on the remote computer;

the data including one or more of: executable instructions contained within or constituted by the object; the size of the object; the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers;

the data being sent in the form of key that is obtained by a hashing process carried out in respect of the object on the remote computer.

This method, which may be carried out by so-called agent software running on the remote computer, allows for efficient sending of data to the base computer, which minimises data transmission and storage requirements and also permits rapid analysis to be made at the base computer.

The key preferably has at least one component that represents executable instructions contained within or constituted by the object.

The key preferably has at least one component that represents data about said object. Said data about said object may include at least one of: the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

The key preferably has at least one component that represents the physical size of the object.

According to a fourth aspect of the present invention, there is provided a method of determining the protection that a remote computer has from malware, the method comprising:

receiving at a base computer details of all or selected security products operating at a point in time on said remote computer;

receiving similar information from other remote computers connected to the base computer; and,

identifying any malware processes that were not identified by said other remote computers having that particular combination of security products.

In this way, the base computer can be used to obtain information as to whether for example a particular, specific combination of operating system and various security products, including settings and signature files existing at a point in time, renders a particular computer having those products and settings susceptible or vulnerable to any particular malware object. The user can be advised accordingly and for example provided with recommendations for remedying the situation.

The method may therefore comprise providing information to the user of said remote computer that said remote computer may be susceptible to attack by said malware processes on the basis of said identifying.

The details of all or selected security products preferably includes the name of the security products, versions, and loaded signature files.

According to a fifth aspect of the present invention, there is provided apparatus for determining the protection that a remote computer has from malware, the apparatus comprising:

a base computer constructed and arranged to receive computer details of all or selected security products operating at a point in time on said remote computer;

the base computer being constructed and arranged to receive similar information from other remote computers connected to the base computer; and,

the base computer being constructed and arranged to identify any malware processes that were not identified by said other remote computers having that particular combination of security products.

According to a sixth aspect of the present invention, there is provided a method of classifying a computer object as malware or as safe, wherein said computer object is a descendant or otherwise related object of a first computer object, the method comprising:

classifying a first computer object as malware or as safe;

identifying in a key relating to said first computer object a component that uniquely identifies the first computer object and that is inherited or otherwise present in the key of a descendant or other related computer object of the first computer object; and,

classifying said computer object as malware or as safe as the case may be on the basis of the unique identifier component being present in the key of said computer object.

This aspect uses the concept of ancestry to enable objects to be marked as malware. For example, any particular process may spawn child processes which are therefore related. The key relating to the first object may be inspected to identify a component that uniquely identifies the first object and that is inherited or otherwise present in the key of a descendant or other related object of the first object.

The method may comprise monitoring activities of said first computer object and reclassifying the first computer object as malware in the case that it was initially classified as safe and subsequently determined to be malware, the method further comprising automatically classifying as malware any computer object that has a key in which said unique identifier component is present.

According to a seventh aspect of the present invention, there is provided apparatus for classifying a computer object as malware or as safe, wherein said computer object is a descendant or otherwise related object of a first computer object, the apparatus comprising:

a computer constructed and arranged to classify a first computer object as malware or as safe;

the computer being constructed and arranged to identify in a key relating to said first computer object a component that uniquely identifies the first computer object and that is inherited or otherwise present in the key of a descendant or other related computer object of the first computer object; and,

the computer being constructed and arranged to classify said computer object as malware or as safe as the case may be on the basis of the unique identifier component being present in the key of said computer object.

According to an eighth aspect of the present invention, there is provided a method of installing software on a computer, the method comprising:

on initiation of installation of software on a computer, providing a computer-generated prompt on the computer to a user to ascertain whether the user authorises the installation; and,

ceasing the installation if a user authorisation is not received, else:

receiving at the computer the user's authorisation to proceed with the installation;

proceeding with the installation;

obtaining data about computer objects that are created or used during the installation;

storing said data at the local computer.

This provides for security when a user is installing new software and is not for example connected to a base computer having a community database of the type mentioned above. In that case, the method, which may be implemented in agent software running on the local computer, allows the user to permit the installation to proceed whilst at the same time gathering data about the objects (such as processes, new files, etc.) that are created during the installation.

Preferably, the locally stored data is referred to during the installation to ensure that all objects created or used during

the installation are part of the installation process, and, if it is found that objects created or used during the installation are not part of the installation process, either or both of: (i) ceasing the installation and (ii) providing a computer-generated prompt on the computer to the user accordingly. This allows the method to ensure that only those objects that are required for the installation are permitted to be created or used and thus avoids unwittingly allowing malware to install (given that malware often creates objects that are not expected as part of a normal installation of new software).

In a preferred embodiment, the method comprises connecting the computer to a community database that is connectable to a plurality of computers, and uploading the stored data to the community database for comparison with similar data provided by other computers.

The method may comprise downloading data about trusted installers to the computer, said data about trusted installers being referred to during the installation such that any objects relating to or created by the trusted installer are automatically authorised to proceed. This facilitates installation of software that is known a priori to be trustworthy.

Said data about trusted installers may be referred to only for a predetermined time period following receipt at the computer of the user's authorisation to proceed with the installation.

The present invention also includes computer programs comprising program instructions for causing a computer to perform any of the methods described above.

Although the embodiments of the invention described with reference to the drawings comprise computer processes performed in computer apparatus and computer apparatus itself, the invention also extends to computer programs, particularly computer programs on or in a carrier, adapted for putting the invention into practice. The program may be in the form of source code, object code, a code intermediate source and object code such as in partially compiled form, or in any other form suitable for use in the implementation of the processes according to the invention. The carrier be any entity or device capable of carrying the program. For example, the carrier may comprise a storage medium, such as a ROM, for example a CD ROM or a semiconductor ROM, or a magnetic recording medium, for example a floppy disk or hard disk. Further, the carrier may be a transmissible carrier such as an electrical or optical signal which may be conveyed via electrical or optical cable or by radio or other means.

Embodiments of the present invention will now be described by way of example with reference to the accompanying drawings, in which:

FIG. **1** shows schematically apparatus in which an embodiment of the present invention may be implemented;

FIG. **2** is a flowchart showing schematically the operation of an example of a method according to an embodiment of the present invention;

FIG. **3** is a flowchart showing schematically the operation of another example of a method according to an embodiment of the present invention; and,

FIG. **4** is a flowchart showing schematically an information obtaining stage.

Referring to FIG. **1**, a computer network is generally shown as being based around a distributed network such as the Internet **1**. The present invention may however be implemented across or use other types of network, such as a LAN. Plural local or "remote" computers **2** are connected via the Internet **1** to a "central" or "base" computer **3**. The computers **2** may each be variously a personal computer, a server of any type, a PDA, mobile phone, an interactive television, or any other device capable of loading and operating computer

objects. An object in this sense may be a computer file, part of a file or a sub-program, macro, web page or any other piece of code to be operated by or on the computer, or any other event whether executed, emulated, simulated or interpreted. An object **4** is shown schematically in the figure and may for example be downloaded to a remote computer **2** via the Internet **1** as shown by lines **5** or applied directly as shown by line **6**.

In one preferred embodiment, the base computer **3** holds a database **7** with which the remote computers **2** can interact when the remote computers **2** run an object **4** to determine whether the object **4** is safe or unsafe. The community database **7** is populated, over time, with information relating to each object run on all of the connected remote computers **2**. As will be discussed further below, data representative of each object **4** preferably takes the form of a so-called signature or key relating to the object and its effects. As will also be discussed further below, the database **7** may further include a mask for the object **4** that sets out the parameters of the object's performance and operation.

Referring now to FIG. **2**, at the start point **21**, a computer object **4** such as a process is run at a remote computer **2**. At step **22**, by operation of local "agent" software running on the remote computer **2**, the operation of the process is hooked so that the agent software can search a local database stored at the remote computer **2** to search for a signature or key representing that particular process, its related objects and/or the event. If the local signature is present, it will indicate either that the process is considered to be safe or will indicate that that process is considered unsafe. An unsafe process might be one that has been found to be malware or to have unforeseen or known unsafe or malevolent results arising from its running. If the signature indicates that the process is safe, then that process or event is allowed by the local agent software on the remote computer **2** to run at step **23**. If the signature indicates that the process is not safe, then the process or event is stopped at step **24**.

It will be understood that there may be more than two states than "safe" or "not-safe" and choices may be given to the user. For example, if an object is considered locally to be not safe, the user may be presented with an option to allow the related process to run nevertheless. It is also possible for different states to be presented to each remote computer **2**. The state can be varied by the central system to take account of the location, status or ownership of the remote computer or timeframe.

If the object is unknown locally, then details of the object are passed over the Internet **1** or other network to the base computer **3** for storing in the community database **7** and preferably for further analysis at the base computer **3**. In that case, the community database **7** is then searched at step **25** for a signature for that object that has already been stored in the community database **7**. The community database **7** is supplied with signatures representative of objects, such as programs or processes, run by each monitored remote computer **2**. In a typical implementation in the field, there may be several thousands or even millions of remote computers **2** connected or connectable to the base computer **3** and so any objects that are newly released upon the Internet **1** or that otherwise are found on any of these remote computers **2** will soon be found and signatures created and sent to the base computer **3** by the respective remote computers **2**.

When the community database **7** is searched for the signature of the object that was not previously known at the remote computer **2** concerned, then if the signature is found and indicates that that object is safe, then a copy of the signature or at least a message that the object is safe is sent to the local

database of the remote computer **2** concerned at step **26** to populate the local database. In this way, the remote computer **2** has this information immediately to hand the next time the object **4** is encountered. A separate message is also passed back to the remote computer **2** to allow the object to run in the current instance.

If the signature is found in the community database **7** and this indicates for some reason that the object is unsafe, then again the signature is copied back to the local database and marked "unsafe" at step **27**, and/or a message is sent to the remote computer **2** so that running of the object is stopped (or it is not allowed to run) and/or the user given an informed choice whether to run it or not.

If after the entire community database **7** has been searched the object is still unknown, then it is assumed that this is an entirely new object which has never been seen before in the field. A signature is therefore created representative of the object at step **28**, or a signature sent by the remote computer **2** is used for this purpose, and this signature is initially marked as bad or unsafe community database **7** at step **29**. The signature is copied to the local database of the remote computer **2** that first ran the object at step **30**. A message may then be passed to the remote computer **2** to instruct the remote computer **2** not to run the object or alternatively the user may be given informed consent as to whether to allow the object to run or not. In addition, a copy of the object itself may be requested at step **31** by the community database **7** from the remote computer **2**.

If the user at the remote computer **2** chooses to run a process that is considered unsafe because it is too new, then that process may be monitored by the remote computer **2** and/or community database **7** and, if no ill effect occurs or is exhibited after a period of time of n days for example, it may then be considered to be safe. Alternatively, the community database **7** may keep a log of each instance of the process which is found by the many remote computers **2** forming part of the network and after a particular number of instances have been recorded, possibly with another particular number of instances or the process being allowed to run and running safely, the signature in the community database **7** may then be marked as safe rather than unsafe. Many other variations of monitoring safety may be done within this concept.

The details of an object **4** that are passed to the base computer **3** are preferably in the form of a signature or "key" that uniquely identifies the object **4**. This is mainly to keep down the data storage and transmission requirements. This key may be formed by a hashing function operating on the object at the remote computer **2**.

The key in the preferred embodiment is specially arranged to have at least three severable components, a first of said components representing executable instructions contained within or constituted by the object, a second of said components representing data about said object, and a third of said components representing the physical size of the object. The data about the object in the second component may be any or all of the other forms of identity such as the file's name, its physical and folder location on disk, its original file name, its creation and modification dates, vendor, product and version and any other information stored within the object, its file header or header held by the remote computer **2** about it; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers. In general, the information provided in the key may include at least one of these elements or any two or more of these elements in any combination.

In one preferred embodiment, a check sum is created for all executable files, such as (but not limited to) .exe and .dll files, which are of the type PE (Portable Executable file as defined by Microsoft). Three types of checksums are generated depending on the nature of the file:

Type 1: five different sections of the file are check summed. These include the import table, a section at the beginning and a section at the end of the code section, and a section at the beginning and a section at the end of the entire file. This type applies to the vast majority of files that are analysed;

Type 2: for old DOS or 16 bit executable files, the entire file is check summed;

Type 3: for files over a certain predefined size, the file is sampled into chunks which are then check summed. For files less than a certain predefined size, the whole file is check summed.

For the check summing process, in principle any technique is possible. The MD5 (Message-Digest algorithm 5) is a widely-used cryptographic hash function that may be used for this purpose.

This allows a core checksum to be generated by viewing only the executable elements of the checksum and making a comparison between two executables that share common executable code.

For the type 1 checksum mentioned above, three signature processes may be used. The first defines the entire file and will change with almost any change to the file's content. The second attempts to define only the processing instructions of the process which changes much less. The third utilises the file's size, which massively reduces the potential of collisions for objects of differing sizes. By tracking the occurrences of all signatures individually appearing with different counterparts, it is possible to identify processes that have been changed or have been created from a common point but that have been edited to perform new, possibly malevolent functionality.

This "meta data" enables current and newly devised heuristics to be run on the data in the community database **7**.

The data stored in the community database **7** provides an extensive corollary of an object's creation, configuration, execution, behaviour, identities and relationships to other objects that either act upon it or are acted upon by it.

The preferred central heuristics use five distinct processes to establish if an object is safe, unsafe or suspicious.

The first of the said processes utilises the singularity or plurality of names, locations, vendor, product and version information captured and correlated from all of the remote computers **2** that have seen the object. By considering the plurality of this information for a single object, a score can be determined which can be used as a measure of the authenticity and/or credibility of the object. Most safe objects tend not to use a large plurality of identifying information or locations. Rules can be established to consider this information in respect of the type of object and its location. For example, temporary files often utilise a plurality of system generated file names which may differ on each remote computer for the same object. Where an object has little plurality, then it provides a reference point to consider its behaviour in comparison to the known behaviours of other objects that have previously used that identifying information. For example, a new object that purports to be a version of notepad.exe can have its behaviour compared with the behaviour of one or more other objects that are also known as notepad.exe. This comparison may be against a single other object or multiple other objects that use the same or even similar identifying information. In this way, new patterns of behaviour can be identified for the new object. Also it allows the preferred embodiment to police an object's behaviour over time to identify new behavioural patterns that may cause an object that was previously consid-

ered safe to have its status reconsidered. Alternatively, the score based on identities may be considered along with scores for other objects or scores created by other processes on this object to be considered in combinations.

The second of the said processes utilises an object's relationship to other objects that act upon it or upon which it acts. For example, analysis can be made of which object created this object, which objects this object created, which objects created a registry key to run or configure this object, which objects were configured by or had registry keys created by this object, etc. In this regard an object is considered to have a relationship based on the event performed by it upon another object, or upon it by another object. This simple 1-to-1 relationship chain provides a complex series of correlation points, allowing ancestral relationships to be considered for any object and its predecessors by event or its issue (i.e.

child and sub-child processes) by event. This allows a score to be developed that describes its relationships and associations with known, unknown, known safe or known bad objects or a combination thereof. Objects that have specific relationships, volumes of relationships or mixes of relationships to one type or another may be judged safe or unsafe accordingly. Alternatively, the relationship-based score may be considered along with other scores to arrive at a determination of safe or unsafe. This data can also be used to deduce a number of factors about objects related directly or via other objects and their behaviours. For example it is possible to deduce how one object's behaviour can be influenced or changed by its association or linkage to another. Consider for example notepad.exe as supplied by Microsoft with the Windows series of operating systems. It has a limited range of functionality and would not be expected therefore to perform a wide variety of events, such as transmitting data to another computer or running other programs etc. However, the behaviour of notepad.exe could be modified by injecting new code into it, such as via dynamic link library injection (DLL injection). In this case notepad.exe would now have new capabilities derived by the code injection or linkage to another object. Using the data that defines the relationships between objects it is possible to deduce that the new behaviours of a program can be attributed to the association with another object. If that new behaviour is malevolent, then it is possible to mark either or all processes as unsafe as appropriate.

The combination of behaviours captured provide a basis to determine if the object is safe or unsafe. Malware typically exhibit certain behaviour and characteristics. For example, malware frequently has a need to self-persist. This manifests itself in the need to automatically restart on system restarts or upon certain events. Creating objects in specific locations to auto restart or trigger execution is a typical characteristic of malware. Replacing core objects of the Windows system environment are another example of typical characteristics of malware. By providing a pattern of behaviour, the determination of objects to be unsafe or safe can be automated. The centralisation of the community data in the community database **7** provides the ability to rapidly assimilate object behaviours, allowing for the rapid identification and determination of malware. Objects may also perform events upon themselves which can be considered in deriving a score.

The third said process involves time and volumes. Relevant data includes when the object was first seen, when it was last seen, how many times it has been seen, how many times it has been seen in a given interval of time, and the increase or decrease of acceleration in it being seen. This information is highly relevant in determining the prevalence of an object in the community of remote computers. A score is developed based on these metrics which can be used to determine if an object is safe, unsafe or too prevalent to allow it to execute or propagate without very thorough examination. In this case, the object can be temporarily held or blocked from executing pending further information about its behaviour or relationships. This score may also be used in combination with scores from other processes. Time is also highly relevant in combination with other information, including but not limited to behaviours and identities. For example in the case of polymorphic or randomly named objects, time is a powerful qualifier. (A polymorphic virus changes its encryption algorithm and the corresponding encryption keys each time it replicates from one computer to another and so can be difficult to detect by conventional measures.) A program that creates other programs can often be considered normal or abnormal based on its activity over time.

The fourth said process considers the behaviour of an object. This allows a score to be developed based on the types of events performed by an object or events performed on it by itself or other objects. The centralised system of the community database **7** allows for an unlimited number of event types and can consider the object performing the event or the object having the event performed upon it, or both. Some event types also relate to external information other than objects, for example a program performing an event to connect with an Internet Chat Relay site, or a program modifying a non-executable file such as the Windows hosts file. The behavioural events of an object, be they as "actor" (i.e. the object doing something to another object) or as "victim" (i.e. the object has something done to it by another object) of any event can be considered in many ways, such as in combination, in sequence, in volume, in presence or absence, or in any combination thereof. The behavioural events in the preferred embodiment may have been provided by a remote computer **2** or from other external sources. The process can consider these in isolation or in combination. Furthermore it is a feature of the preferred embodiment that the behavioural events can be considered in combination with the status of other objects upon which the object acts or that act upon the object. For example, creating a program may have a different score if the program being created is safe, unsafe, new, unknown or suspicious. Similarly, a program that is created by a known bad program will likely have a different score attributed to its creation event depending on the status of the object creating it.

The fifth process considers the behaviour of a web page or script. In this respect, the web page and url combination is assigned a unique identity which allows its behaviour to be tracked as if it were an object like any other executable file. In this example, the web page may perform events that would normally be seen as events performed by the web browser (e.g. IExplore.exe or Firefox.exe). The preferred embodiment substitutes the identifying details and signatures of the web browser for the "pseudo" object identity associated with the web page being displayed or executing within the browser. In this respect, the status of the web page and/or web site to which it relates may be determined as safe, unsafe, unknown or suspicious in the same way as any other object. The web page's "pseudo" object identity also allows the preferred embodiment to block, interject or limit the functionality of that web page or web site to prevent some or all of its potentially unsafe behaviour or to provide the remote user with qualifying information to guide them about the safety of the web site, web page or their content.

Amongst other types, the types of meta data captured might be:

"Events": these define the actions or behaviours of an object acting upon another object or some other entity. The event has three principal components: the key of the object performing the act (the "Actor"), the act being performed (the "Event Type"), and the key of the object or identity of an other entity upon which the act is being performed (the "Victim"). While simple, this structure allows a limitless series of behaviours and relationships to be defined. Examples of the three components of an event might be:

Actor	Event Type	Victim
Object 1	Creates Program	Object 2
Object 1	Sends data	IP Address 3
Object 1	Deletes Program	Object 4
Object 1	Executes	Object 2
Object 2	Creates registry key	Object 4

"Identities": these define the attributes of an object. They include items such as the file's name, its physical location on the disk or in memory, its logical location on the disk within the file system (its path), the file's header details which include when the file was created, when it was last accessed, when it was last modified, the information stored as the vendor, the product it is part of and the version number of the file and it contents, its original file name, and its file size.

"Genesisactor"—the key of an object that is not the direct Actor of an event but which is the ultimate parent of the event being performed. For example in the case of a software installation, this would be the key of the object that the user or system first executed and that initiated the software installation process, e.g. Setup.exe.

"Ancillary data": many events may require ancillary data, for example an event such as that used to record the creation of a registry run key. In this situation the "event" would identify the Actor object creating the registry run key, the event type itself (e.g. "regrunkey"), and the Victim or subject of the registry run key. The ancillary data in this case would define the run key entry itself; the Hive, Key name and Value.

"Event Checksums": because the event data can be quite large extending to several hundred bytes of information for a single event, its identities for the Actor and Victim and any ancillary data, the system allows for this data itself to be summarised by the Event Checksums. Two event checksums are used utilising a variety of algorithms, such as CRC and Adler. The checksums are of the core data for an event. This allows the remote computer **2** to send the checksums of the data to the central computer **3** which may already have the data relating to those checksums stored. In this case, it does not require further information from the remote computer **2**. Only if the central computer **3** has never received the checksums will it request the associated data from the remote computer **2**. This affords a considerable improvement in performance for both the remote and central computers **2**, **3** allowing much more effective scaling.

Thus, the meta data derived from the remote computers **2** can be used at the community database **7** to define the behaviour of a process across the community. As mentioned, the data may include at least one of the elements mentioned above (file size, location, etc.) or two or three or four or five or six or all seven (or more elements not specifically mentioned here). This may be used accordingly to model, test and create new automated rules for use in the community database **7** and as rules that may be added to those held and used in the local database of the remote computers **2** to identify and determine the response of the remote computers **2** to new or unknown processes and process activity.

Moreover, it is possible to monitor a process along with any optional sub-processes as an homogenous entity and then compare the activities of the top level process throughout the community and deduce that certain, potentially malevolent practices only occur when one or more specific sub-processes are also loaded. This allows effective monitoring (without unnecessary blocking) of programs, such as Internet Explorer or other browsers, whose functionality may be easily altered by downloadable optional code that users acquire from the Internet, which is of course the principal source of malevolent code today.

The potentially high volume of active users gives a high probability of at least one of them being infected by new malware. The speed of propagation can be detected and recorded so that the propagation of malware can be detected and malware designated as bad on the basis of the speed of propagation, optionally in combination with the other factors discussed above, such as file size, location and name. The simple volume of infection can also be used as a trigger. In a further embodiment, difference of naming of an otherwise identical piece of code combined with acceleration of first attempts to execute the code within the community allows pattern matching that will show up an otherwise identically signatured piece of code as bad.

This feature allows the statement in some embodiments that "nothing will propagate in our community faster than X without being locked down", so that if any process or event propagates more quickly over a given duration, it is marked as bad. This is for reasons of safety given that if for example an object is propagating quickly enough, then it might infect computers before it can be analysed to determine whether or not it is malware.

This process can be automated by the identification of the vector of propagation in the community (i.e. the source of type of propagation), from timestamp data held in the community database and the marking of a piece of code that has these attributes as bad. By comparison, it is believed that all other anti-malware providers rely on a simplistic known bad model and therefore are reliant primarily on malware infection actually occurring on terminals and being reported.

Thus, the community database **7** can be used to make early diagnosis, or simply to take precautionary measures, and thus stop potentially fast propagating worms and other malware very, very early in their life cycle. Given that it is possible to create a worm that can infect every computer connected to the Internet within a matter of a few minutes, this feature is highly desirable.

Even faster determination may be made by combining data defining the speed of propagation of a new piece of software with metadata collected by the agent software from the remote computers **2** and fed to the community database **7**. This includes monitoring processes that attempt to conceal themselves from the user by randomly changing name and/or location on the remote computers **2**. It also includes a process's attempt to create an identical copy (i.e. with identical code contents) on the computer but with a different name. This is a classic attribute of a worm.

The signature of an object may comprise or be associated with a mask which can be built up with use of that object and which indicates the particular types of behaviour to be expected from the object. If an object is allowed to run on a remote computer **2**, even if the initial signature search **22** indicates that the object is safe, then operation of that object may be monitored within the parameters of the mask. The

mask might indicate for example, the expected behaviour of the object; any external requests or Internet connections that that object might legitimately have to make or call upon the remote computer **2** to make, including details of any ports or interfaces that might be required to be opened to allow such communication; any databases, either local or over a local area network or wide area network or Internet, that may be expected to be interrogated by that object; and so on. Thus, the mask can give an overall picture of the expected "normal" behaviour of that object.

In practice, therefore, in one embodiment the behaviour of the object is continually monitored at the remote computer(s) **2** and information relating to that object continually sent to and from the community database **7** to determine whether the object is running within its expected mask. Any behaviour that extends beyond the mask is identified and can be used to continually assess whether the object continues to be safe or not. Thus, if for example the object, on a regular basis (say monthly or yearly) opens a new port to update itself or to obtain regular data, then this information is flagged. If it is found that the object has done this on other remote computers and has had no ill effects, or this behaviour is known from other objects and known to be safe, then this behaviour might be considered as safe behaviour and the mask is then modified to allow for this. If it has been found previously that this new behaviour in fact causes unsafe or malevolent results, then the object can then be marked as unsafe even if previously it was considered safe. Similarly, if the object attempts to connect to a known unsafe website, database or to take action that is known as generally being action only taken by unsafe programs, then again the object may be considered to be unsafe.

This is shown schematically in FIG. **3**. FIG. **3** also shows the concept that any object can be pre-authorised by, for example a trusted partner, such as a major software company, a verification authority, a Government department, and so on. Pre-authorisation enables a supplier of a new object, which has not been released before, to get pre-authorisation for that object, and optionally includes the provision by that supplier of a mask detailing the expected and allowable behaviour of that object.

Referring to FIG. **3** for example, when a process is run, the local and/or community databases are searched as before at step **31**. If the process is not a pre-authorised one, then the steps of FIG. **2** may be taken and the process might be allowed to run or not at step **32**. If the process is pre-authorised, as determined at step **33**, then it is immediately allowed to run, step **34**. This may terminate the operation of the method. However, in a preferred variation, the process is then monitored whilst running, and is monitored each time it is run in the future in a monitoring state step **35** to determine whether its behaviour falls within its pre-authorised mask **36**. If the behaviour falls within the pre-authorised behaviour, then the process is allowed to continue to run. If the behaviour extends beyond the allowed mask, such as by trying to instigate further processes or connections that have not been pre-authorised, then this behaviour is flagged at an alert step **37**. Various actions could be taken at this stage. The process might simply not be allowed to run. Alternatively, the trusted authority that initially enabled pre-authorisation might be contacted, who may be able to confirm that this behaviour is acceptable or not. If it is acceptable, then the mask could be modified accordingly. If not acceptable, then the process might be marked as unsafe. Many other actions may be taken upon the noting of such an alert state.

If the process has been found not to be pre-authorised at step **33** but is nevertheless allowed to run, then the process is monitored at step **38** in order to generate a mask **39** representative of the normal behaviour of that process. Data representative of this mask might be sent to the community database **7** for scanning when other computers run that process. By continually monitoring a process each time it is run or during running of the process, any behaviour that differs from previous behaviour of the process can be noted and the mask can be modified, or this behaviour might be used to determine that a process that was once considered safe should now be considered to be unsafe.

In another embodiment, a computer **2** may have agent software installed that periodically or on-demand provides information to the community database **7** that is representative of all or selected ones of the software products loaded on or available the computer **2**. In particular, this may be information on one or more of: all the locally-loaded security products (such as anti-malware systems including anti-virus software, anti-spyware, anti-adware and so on), firewall products, specific settings and details of which signature files are currently loaded, version details for the operating system and other software, and also information such as which files are operating and the particular version and software settings at any time. (It will be understood from the following that auditing and testing for a match for more of these criteria increases the likelihood of computers being very similarly arranged and thus reduces the rate of false negatives and positives during the match search.)

The information relating to these software products, etc. may be provided individually to the community database. Preferably however, again for reasons of data quantity for storage and transmission, the information is provided as a signature or key representative of the information (e.g. by a hashing or compression function at the computer **2**). FIG. **4** shows schematically how the details of all local security products, versions, signature files, firewall settings, etc. **40** are used to create a key **41**. The key is transmitted to the community database **7**. Since the community database **7** is provided with such information from many, possibly millions, of users' computers **2**, it is likely to hold corresponding information for other computers **2** that have the same or a similar configuration of security products, etc. Thus, the community database **7** can be searched at step **42** for other computers **2** having the same or a similar combination of security products including the same setting, signature files loaded and so on.

The community database **7** in this embodiment is also provided by the agent software with details of processes run by every computer **2** and thus knows whether or not a process has been detected by each computer **2**.

In this way, the community database **7** can be used to obtain information as to whether for example a particular, specific combination of operating system and various security products, including settings and signature files existing at a point in time, renders a particular computer **2** having those products and settings susceptible or vulnerable to any particular malware object.

In a simple example, if for example the database knows that a computer in the past has version A of anti-virus product B with downloaded signature update C, and also has a firewall D with particular settings E, and perhaps anti-spyware software F with signature updates G, but that a particular malware process P was not detected by this combination of programs at that point in time, then this information can be provided to a computer **2** that is known as having that combination of security programs/settings, and can be used to indicate that that computer **2** is vulnerable in the short term to attack by that particular malware process. This information can be presented to the user either by displaying a window **43** on the

screen display of the computer **2** or by directing the user to a particular website which explains the position in more detail. The user might be informed for example that their particular combination of security products, etc., exposes their computer to a risk of being infected by the Sobig virus as that virus is not detectable by their computer. The user might be offered specific advice (e.g. to update a particular anti-virus program with a particular signature file) or software to download and install to remove the risk.

Thus, the community database **7**, when provided with information relating to all the security products, etc. on a particular computer at a particular time, is searched for events for processes marked as "bad" that occurred on computers with that particular mix of security products and that were not locally detected. This information can then be fed back to the user of the particular computer, for example directly or by directing the user to a website. This information can be provided virtually in real-time, allowing a new user or a user of a new computer to be able to increase the computer's effective security very quickly.

The preferred method also tracks which objects are related to each other and uses the concept of ancestry to enable objects to be marked as malware. For example, any particular process may spawn child processes which are therefore related. The key relating to the first object may be inspected to identify a component that uniquely identifies the first object and that is inherited or otherwise present in the key of a descendant or other related object of the first object. This component is referred to herein as a "gene". This general technique may be used in a number of ways:

a) A known and trusted parent process is afforded the ability to create child processes which may be automatically marked as safe to run on the local computer. It is also possible that this "inherited" property may be passed down to grand children processes and so on. This safe status is passed to the parent's child processes and possibly, through them, further child processes (referred to here as "issue"), such signatures for the issue can all automatically be recorded in the local database as good. This allows the issue processes to be quickly marked as good, even if a connection to the community database **7** is not available.

b) By monitoring activity of the parent process, if it is later found that the parent process is malware, then all of the issue processes can all automatically be recorded in the local database as bad.

C) Similarly, by monitoring activity of the issue processes, if it is later found that one of the issue processes is malware, then one or more of the parent process and all of the other issue processes (i.e. all of the related processes in this context) can all automatically be recorded in the local database as bad.

d) Parental creation of a signature for a child or children including the ability for these to be automatically marked as either good or bad depending on the parent's behaviour and determination. Note that in some embodiments the product can "watch" or monitor the birth of a child process and automatically create the signature upon arrival. This provides the ability to monitor the creation of a bad program by another bad program. It is possible therefore to monitor the ancestry of a program so if for example the grandfather creates a program (the father) and this in turn creates a bad program (the son), it is possible automatically to determine the father as a bad program.

e) A feature may be included that allows for automatic forfeiture of a child's inherited ability to trigger the automatic creation of signatures on any further births because the child, as parent, has already produced bad offspring. Preferably, a rule is that if a file has one bad offspring then the inherited ability can be automatically removed.

f) An ability to watch clones or identical twins of objects (e.g. the same process running on other systems in the community) to compare the pattern of their issue and to make decisions as to whether or not to treat any particular process as malware.

One or more of these features a) to f) can be used to provide a solution to the problem of producing a security product that can be used effectively without 100% reliance on being permanently connected to the Internet, which is often impractical. Examples of this are Windows Update and other processes used more and more by vendors who wish to be able to roll out product updates automatically across the worldwide web.

Possible benefits of these types of features above conventional software are as follows. Antivirus software tends to have a cache of known bad signatures. The problem is keeping this up to date. Take the simple example of someone buying a new computer. The computer comes with an antivirus product preloaded with a signature cache. Between the time when the PC was built, shipped to the store and bought by the user several days or weeks will have passed. The user starts the PC and is exposed to any new virus or malware which was created after the PC was built. Full protection requires the user to connect to the internet and download updates. This cannot be guaranteed to occur ahead of other activities by the user on the internet (almost physically impossible to guarantee). With a local cache of known good processes, as in the embodiments of the present invention, it is possible to ship the computer/terminal preloaded with a pre-generated cache of signatures for all of the good (clean) software preloaded by the computer manufacturer. In this case the user can connect to the internet knowing that any new or updated programs will be immediately detected and verified. Also any auto-updating software can function forcing signatures to be automatically built for its children and more remote off-spring (i.e. grandchildren, great-grandchildren, etc).

Reference is now made to "Defeating Polymorphism: Beyond Emulation" by Adrian Stepan of Microsoft Corporation published in "Virus Bulletin Conference, October 2005" and also to U.S. 60/789,156 filed on 5 Apr. 2006, the entire content of which are hereby incorporated by reference. In that paper and patent application, there are disclosed methods of decrypting files to allow the files to be analysed to determine whether or not the file actually is malware. In U.S. 60/789,156 in particular, there is disclosed a method of unpacking/decrypting an executable computer file using a host computer, the method comprising: partitioning the executable computer file into plural basic blocks of code; translating at least some of the basic blocks of code into translated basic blocks of code that can be executed by the host computer; linking at least some of the translated basic blocks of code in memory of the host computer; and, executing at least some of the translated basic blocks of code on the host computer so as to enable the executable computer file to be unpacked or decrypted, whereupon the unpacked or decrypted executable computer file can be analyzed to determine whether the executable computer file is or should be classed as malware. There is also disclosed in U.S. 60/789,156 a method of unpacking/decrypting an executable computer file, the method comprising: partitioning the executable computer file into plural basic blocks of code; creating at least a read page of cache memory for at least some of the basic blocks, the read page of cache memory storing a read cached real address corresponding to a read cached virtual memory address for the respective basic block,

and creating at least a write page of cache memory for at least some of the basic blocks, the write page of cache memory storing a write cached real address corresponding to a write cached virtual memory address for the respective basic block; and, emulating the executable file by executing at least some of the basic blocks of code so as to enable the executable computer file to be unpacked or decrypted, whereupon the unpacked or decrypted executable computer file can be analyzed to determine whether the executable computer file is or should be classed as malware; wherein during the execution of a basic block, at least one of the read page and the write page of cache memory is checked for a cached real address corresponding to the virtual address that is being accessed for said basic block.

The techniques disclosed in these documents can be used in the present context when it is desired to analyse a file in detail. More generally however, the techniques disclosed in these papers, and particularly the enhanced techniques disclosed in U.S. 60/789,156, can be used to provide information about the activity of a file when it is run on a computer because the techniques disclosed in these papers emulate the running of the file and therefore allow the file's activity to be interpreted.

A further situation arises when users wish to install software while offline. In a preferred embodiment, when a user attempts to install new software while offline, agent software running on the user's computer **2** prompts the user for authorisation to allow the installation process to proceed such that the execution of the installation can itself be "authorised" by the user. This authorisation by the user is treated as a "Genesis" event and will be so termed hereinafter. However there are some processes commonly used in installation of software that communicate with other existing programs on the installation machine, e.g. Microsoft's MSIEXEC.

The Genesis approach involves a process that generates signatures as a result of the user's authorisation on the user's computer **2**. Those signatures are stored in the local database on the user's computer **2**. In one embodiment, those locally stored signatures are referred to as necessary by the agent software during the installation process so that the installation can proceed. Alternatively, the security checks made by the agent software can be switched off during the installation process. The switching off may only be for a limited duration, such as a few minutes which should be sufficient to allow most software to be installed, the off time optionally being user-configurable.

In any event, once the installation has been completed and the user's computer **2** connected to the community database **7**, the agent software on the user's computer **2** can upload the signatures relating to the installation from the user's local database to the community database **7**. With corresponding data from other users' computers **2**, the community database **7** can then be used to make a rapid determination that the installation of this particular software is benign.

As a variant to this embodiment, when a user's computer **2** is at some point in time on-line, the agent software on the user's computer **2** may download signatures of a so-called "trusted installer" or "licensed installer". This allows the operation of a method such that a "licensed installer" and any child processes of the licensed installer are permitted to execute while a Genesis event is "current", e.g. within a period of minutes after an authorisation from the user. Preferably signatures of licensed installers are always downloaded, as and when added to the community database **7**, to a remote computer **2** while online.

There may be further refinements to this method, such as to prevent licensed installers executing if a media change has occurred during the currency of a Genesis event. However, any unknown processes, which may be malware, may still be detected and blocked. Having a small number of licensed installers facilitates download, as minimal data is required, especially compared to downloading virus signature files. As another example, super-trusted installers, such as for example "Windows Update", may be employed whereby all new processes created by the super-trusted installer are marked immediately as safe.

In the case of an installation occurring when the remote computer **2** is connected to the community database **7**, another option is that if the software that is about to be installed is not known to the community database **7**, then the system will block the installation or alert the user. For example, a message may be displayed at the user's computer **2** to the effect that "You are about to install some software. This software is not known to [the community]. Are you sure you wish to proceed?".

Embodiments of the present invention have been described with particular reference to the examples illustrated. However, it will be appreciated that variations and modifications may be made to the examples described within the scope of the present invention.

The invention claimed is:

1. A method of classifying a computer object as malware, the method comprising:

at a base computer, receiving data about a computer object from a first remote computer on which the computer object or similar computer objects are stored, wherein said data includes information about events initiated or involving the computer object when the computer object is created, configured or runs on the first remote computer, said information including at least an identity of an object initiating the event, the event type, and an identity of an object or other entity on which the event is being performed;

at the base computer, receiving data about the computer object from a second remote computer on which the computer object or similar computer objects are stored, wherein said data includes information about events initiated or involving the computer object when the computer object is created, configured, or runs on the second remote computer, said information including at least an identity of an object initiating the event, the event type, and an identity of an object or other entity on which the event is being performed;

storing, at the base computer, said data received from the first and second remote computers;

correlating, by the base computer, at least a portion of the data about the computer object received from the first remote computer to at least a portion of the data about the computer object received from the second remote computer;

comparing, by the base computer, the correlated data about the computer object received from the first and second remote computers to other objects or entities to identify relationships between the correlated data and the other objects or entities; and

classifying, by the base computer, the computer object as malware on the basis of said comparison.

2. A method according to claim **1**, wherein the relationship is between objects related directly, or between objects related via other objects.

3. A method according to claim **1**, wherein where an object and other objects are found to have a relationship, monitoring

those related objects as a homogenous entity and identifying a common object of the homogenous entity as a malware object.

4. A method according to claim **1**, comprising deducing an object to be malware based on the behaviour of a related object.

5. A method according to claim **1**, wherein if at least one other object to which said computer object is related is classed as malware, then classifying said computer object as malware.

6. A method according to claim **1**, wherein said other objects include the object or similar objects stored on at least one of the first and second remote computers.

7. A method according to claim **1**, wherein said other objects include other objects that are parent objects or child objects or otherwise process-related objects to said computer object.

8. A method according to claim **1**, wherein the data about the computer object that is sent from the respective remote computer to the base computer and that is used in the comparison includes one or more of:

executable instructions contained within or constituted by the computer object;

the size of the computer object;

the current name of the computer object;

the physical and folder location of the computer object on disk;

the original name of the computer object;

the creation and modification dates of the computer object;

vendor, product and version and any other information stored within the computer object; and

the computer object header or header held by the respective remote computer.

9. A method according to claim **1**, wherein the data is sent in the form of a key that is obtained by a hashing process carried out in respect of the objects on the respective remote computer.

10. A method according to claim **9**, wherein the key has at least one component that represents executable instructions contained within or constituted by the computer object.

11. A method according to claim **9**, wherein the key has at least one component that represents data about said computer object.

12. A method according to claim **11**, wherein said data about said computer object includes at least one of:

the current name of the computer object;

the physical and folder location of the computer object on disk;

the original name of the computer object;

the creation and modification dates of the computer object;

vendor, product and version and any other information stored within the computer object;

the computer object header or header held by the respective remote computer; and,

events initiated by or involving the computer object when the computer object is created, configured or runs on the respective remote computers.

13. A method according to claim **9**, wherein the key has at least one component that represents the physical size of the computer object.

14. A method according to claim **1**, comprising initially classifying a computer object as not malware, generating a mask for said computer object that defines acceptable behaviour for the computer object, monitoring an operation of the computer object on at least one of the first and second remote computers and reclassifying the computer object as malware if the actual monitored behaviour extends beyond that permitted by the mask.

15. An apparatus for classifying a computer object as malware, the apparatus comprising:

a base computer constructed and arranged to receive data about a computer object from a first remote computer on which the computer object or similar computer objects are stored, wherein said data includes information about events initiated or involving the computer object when the computer object is created, configured, or runs on the first remote computer, said information including at least an identity of an object initiating the event, the event type, and an identity of an object or other entity on which the event is being performed;

the base computer being constructed and arranged to receive data about the computer object from a second remote computer on which the computer object or similar computer objects re stored, wherein said data includes information about events initiated or involving the computer object when the computer object is created, configured, or runs on the second remote computer, said information including at least an identity of an object initiating the event, the event type, and an identity of an object or other entity on which the event is being performed;

the base computer being constructed and arranged to correlate at least a portion of the data about the computer object received from the first remote computer to at least a portion of the data about the computer object received from the second remote computer;

the base computer being constructed and arranged to compare the data about the computer object received from the first and second remote computers to other objects or entities to identify relationships between the correlated data and the other objects or entities; and

the base computer being constructed and arranged to classify the computer object as malware on the basis of said comparison.

16. Tha apparatus according to claim **15**, wherein the base computer is constructed and arranged so as to identify the relationship between objects related directly, or between objects related via other objects.

17. The apparatus according to claim **15**, wherein the base computer is constructed and arranged so as, where an object and other objects are found to have a relationship, to monitor those related objects as a homogenous entity and identify a common object of the homogenous entity as a malware object.

18. The apparatus according to claim **15**, wherein the base computer is constructed and arranged so as to deduce an object to be malware based on the behaviour of a related object.

19. The apparatus according to claim **15**, wherein the base computer is constructed and arranged so that if at least one other object to which said object is related is classed as malware, then said object is classified as malware.

20. The apparatus according to claim **15**, wherein the base computer is constructed and arranged so that said other objects include the object or similar objects stored on at least one of the first and second remote computers.

21. The apparatus according to claim **15**, wherein the base computer is constructed and arranged so that said other objects include other objects that are parent objects or child objects or otherwise process-related objects to said object.

22. The apparatus according to claim **15**, wherein the base computer is constructed and arranged so as to compare the data where the data includes one or more of:

executable instructions contained within or constituted by the computer object;

the size of the computer object;

the current name of the computer object;

the physical and folder location of the computer object on disk;

the original name of the computer object;

the creation and modification dates of the computer object;

vendor, product, and version and any other information stored within the computer object; and

the computer object header or header held by the respective remote computer.

23. The apparatus according to claim **15**, wherein the base computer is constructed and arranged so as to be able to process data that is sent in the form of a key that is obtained by a hashing process carried out in respect of the objects on said respective remote computer.

24. The apparatus according to claim **23**, wherein the key has at least one component that represents executable instructions contained within or constituted by the computer object.

25. The apparatus according to claim **23**, wherein the key has at least one component that represents data about said computer object.

26. Apparatus according to claim **25**, wherein said data about said object includes at least one of:

the current name of the computer object;

the physical and folder location of the computer object on disk;

the original name of the comptuer object;

the creation and modification dates of the computer object;

vendor, product, and version and any other information stored within the computer object;

the computer object header or header held by the respective remote computer; and

events initiated by or involving the computer object when the computer object is created, configured, or runs on the respective remote computers.

27. The apparatus according to claim **23**, wherein the key has at least one component that represents the physical size of the computer object.

28. The apparatus according to claim **15**, wherein the base computer is constructed and arranged to generate a mask for a computer object that is initially classed not as malware, said mask defining acceptable behaviour for the computer object, and the base computer is constructed and arranged to monitor an operation of the computer object on at least one of the first and second remote computers and to reclassify the computer object as malware if the actual monitored behaviour extends beyond that permitted by the mask.

29. A computer program recorded on non-transitory computer readable medium comprising program instructions for causing a computer to perform the method of claim **1**.

30. A method according to claim **1**, wherein the data received about the computer object from the first remote computer is different than the data received about the computer object from the second remote computer.

* * * * *

Exhibit 3

U 8164950

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

October 07, 2021

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THIS OFFICE OF:

U.S. PATENT: *9,578,045*
ISSUE DATE: *February 21, 2017*

By Authority of the
Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office

R GLOVER
Certifying Officer

US009578045B2

(12) **United States Patent**
Jaroch et al.

(10) **Patent No.: US 9,578,045 B2**
(45) **Date of Patent: Feb. 21, 2017**

(54) **METHOD AND APPARATUS FOR PROVIDING FORENSIC VISIBILITY INTO SYSTEMS AND NETWORKS**

(71) Applicant: **Webroot Inc.**, Broomfield, CO (US)

(72) Inventors: **Joseph Jaroch**, Deer Park, IL (US); **Jacques Etienne Erasmus**, Belper (GB); **Paul Barnes**, Derby (GB); **Johannes Mayr**, Linz (AT); **Michael Leidesdorff**, Niwot, CO (US); **Marco Giuliani**, Bastia Umbra (IT); **Christopher Jon Williams**, Derbyshire (GB); **Chad Edward Bacher**, Arvada, CO (US)

(73) Assignee: **WEBROOT INC.**, Broomfield, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **14/270,069**

(22) Filed: **May 5, 2014**

(65) **Prior Publication Data**

US 2014/0331322 A1 Nov. 6, 2014

Related U.S. Application Data

(60) Provisional application No. 61/819,470, filed on May 3, 2013.

(51) **Int. Cl.**
G06F 21/00 (2013.01)
H04L 29/06 (2006.01)

(52) **U.S. Cl.**
CPC ***H04L 63/1416*** (2013.01)

(58) **Field of Classification Search**
CPC combination set(s) only.
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2013/0298244 A1* 11/2013 Kumar G06F 21/52 726/25

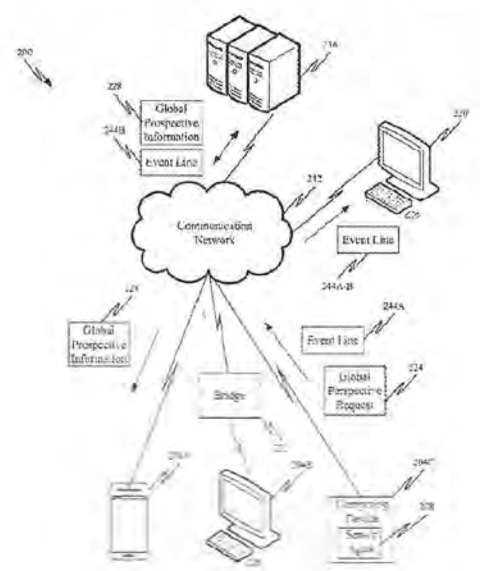
* cited by examiner

Primary Examiner — Ghazal Shehni
(74) *Attorney, Agent, or Firm* — Merchant & Gould P.C.

(57) **ABSTRACT**

Methods and systems for providing forensic visibility into systems and networks are provided. More particularly, a sensor agent may receive events defining an action of a first object acting on a target. The object, the event, and the target are then correlated to at least one originating object such that an audit trail for each individual event is created. A global perspective indicating an age, popularity, a determination as to whether the object may be malware, and IP/URL information associated with the event may then be applied to at least one of the object, the event, the target, and the originating object. A priority may then be determined and assigned to the event based on at least the global perspective. An event line containing event information is then transmitted to an end recipient where the information may be heuristically displayed.

19 Claims, 15 Drawing Sheets



Copy provided by USPTO from the PIRS Image Database on 10-05-2021

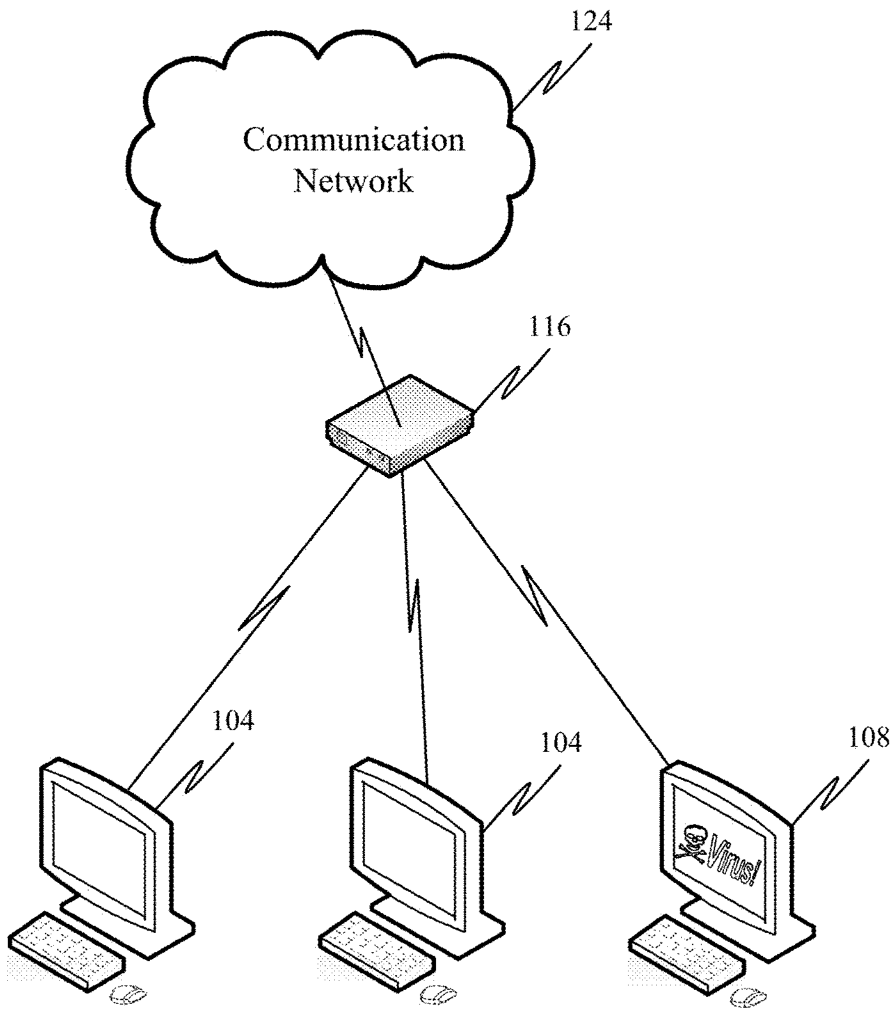


Figure 1

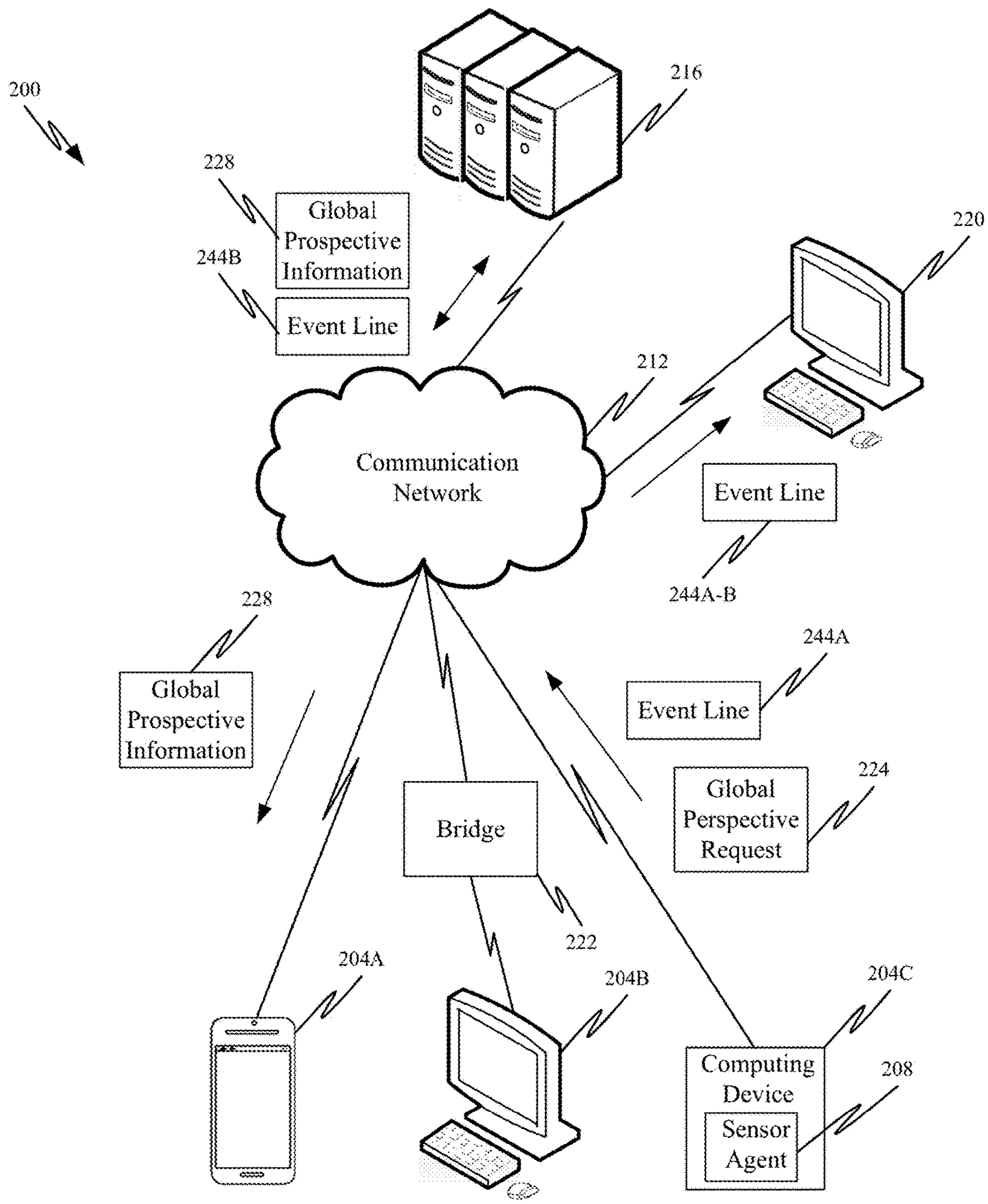
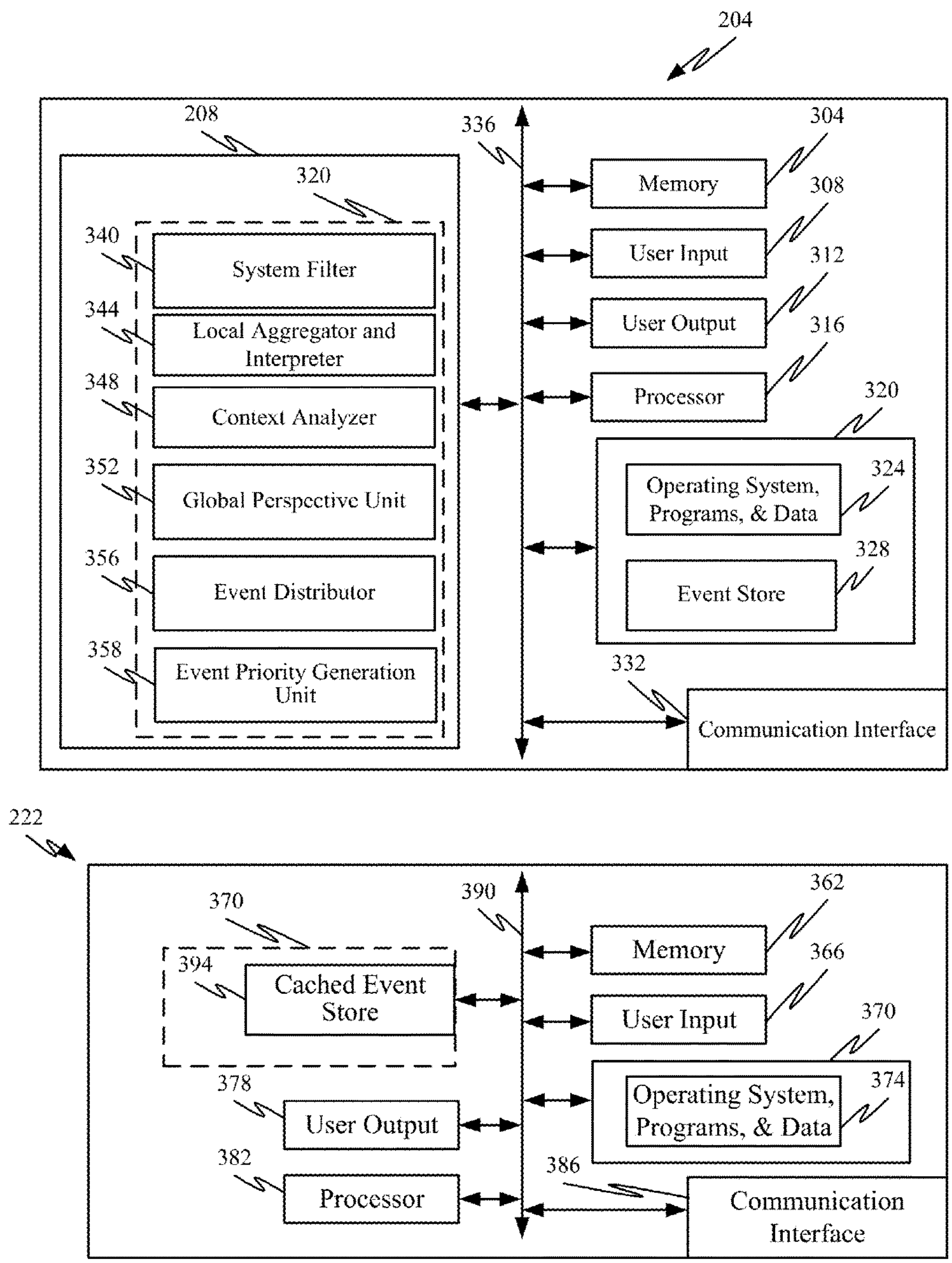

200
216
228
Global Prospective Information
244B
Event Line
220
212
Communication Network
Event Line
244A-B
228
Global Prospective Information
244A
Event Line
224
Global Perspective Request
Bridge
222
204A
204B
204C
208
Computing Device
Sensor Agent

Figure 2



204
208
336
304
320
Memory
308
340
System Filter
User Input
312
344
Local Aggregator and Interpreter
User Output
316
348
Context Analyzer
Processor
320
352
Global Perspective Unit
Operating System, Programs, & Data
324
356
Event Distributor
328
Event Store
358
Event Priority Generation Unit
332
Communication Interface
222
370
390
362
Memory
394
Cached Event Store
366
User Input
370
378
User Output
Operating System, Programs, & Data
374
382
Processor
386
Communication Interface

Figure 3

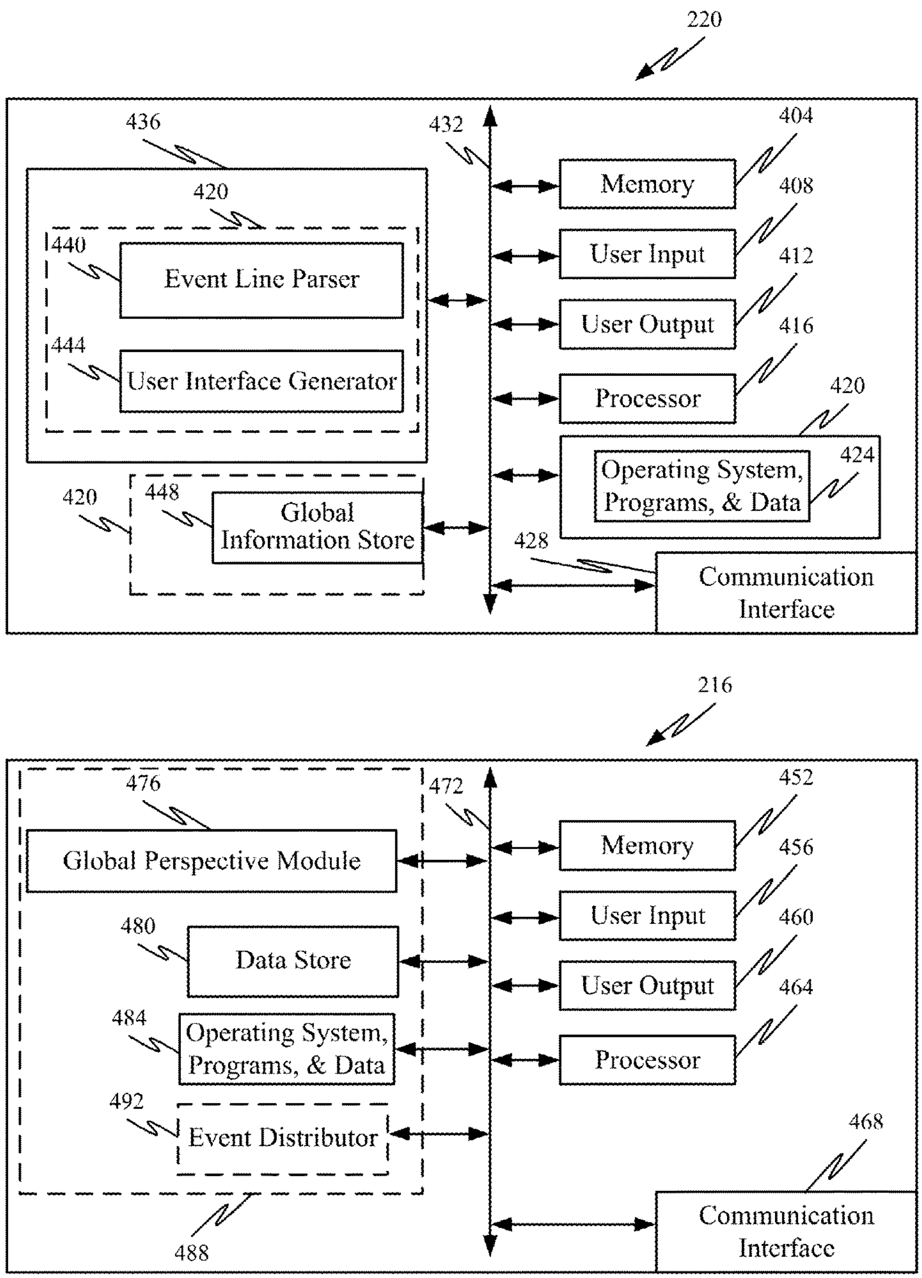

220
436
432
404
Memory
420
408
440
Event Line Parser
User Input
412
User Output
444
416
User Interface Generator
Processor
420
Operating System, Programs, & Data
424
420
448
Global Information Store
428
Communication Interface
216
476
472
452
Global Perspective Module
Memory
456
User Input
480
460
Data Store
User Output
484
464
Operating System, Programs, & Data
Processor
492
468
Event Distributor
Communication Interface
488

Figure 4

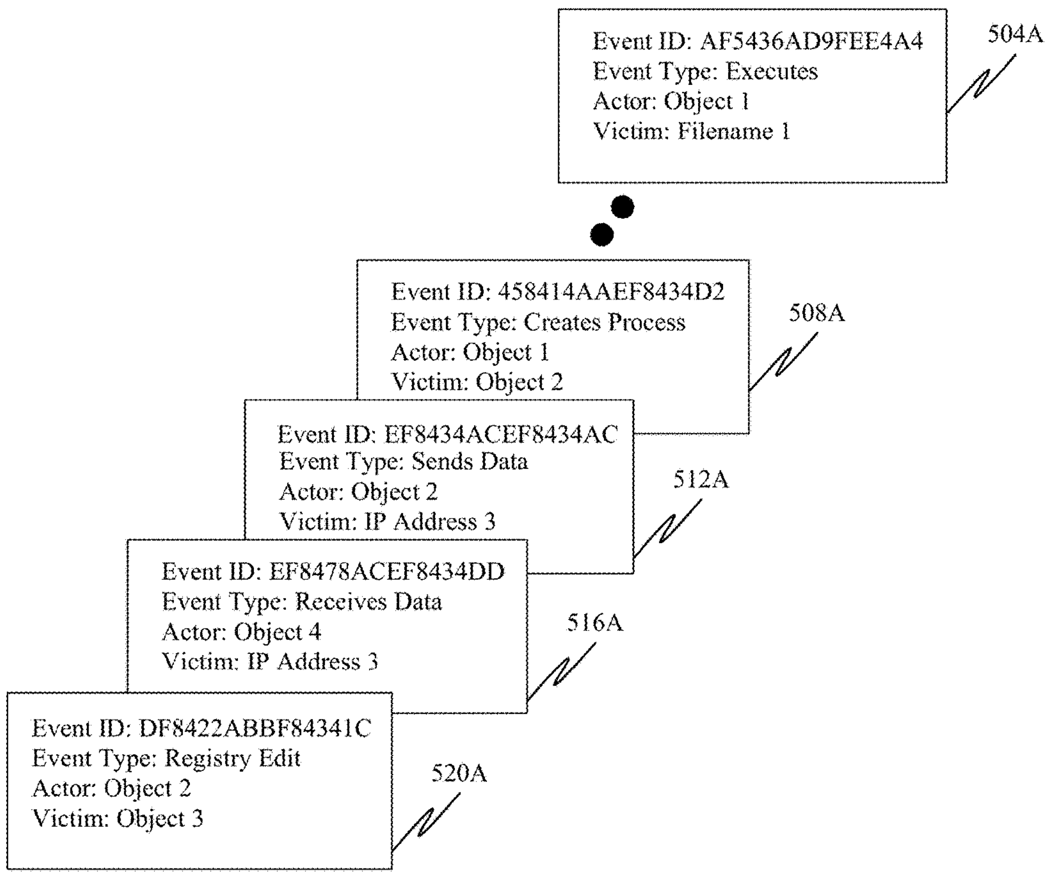
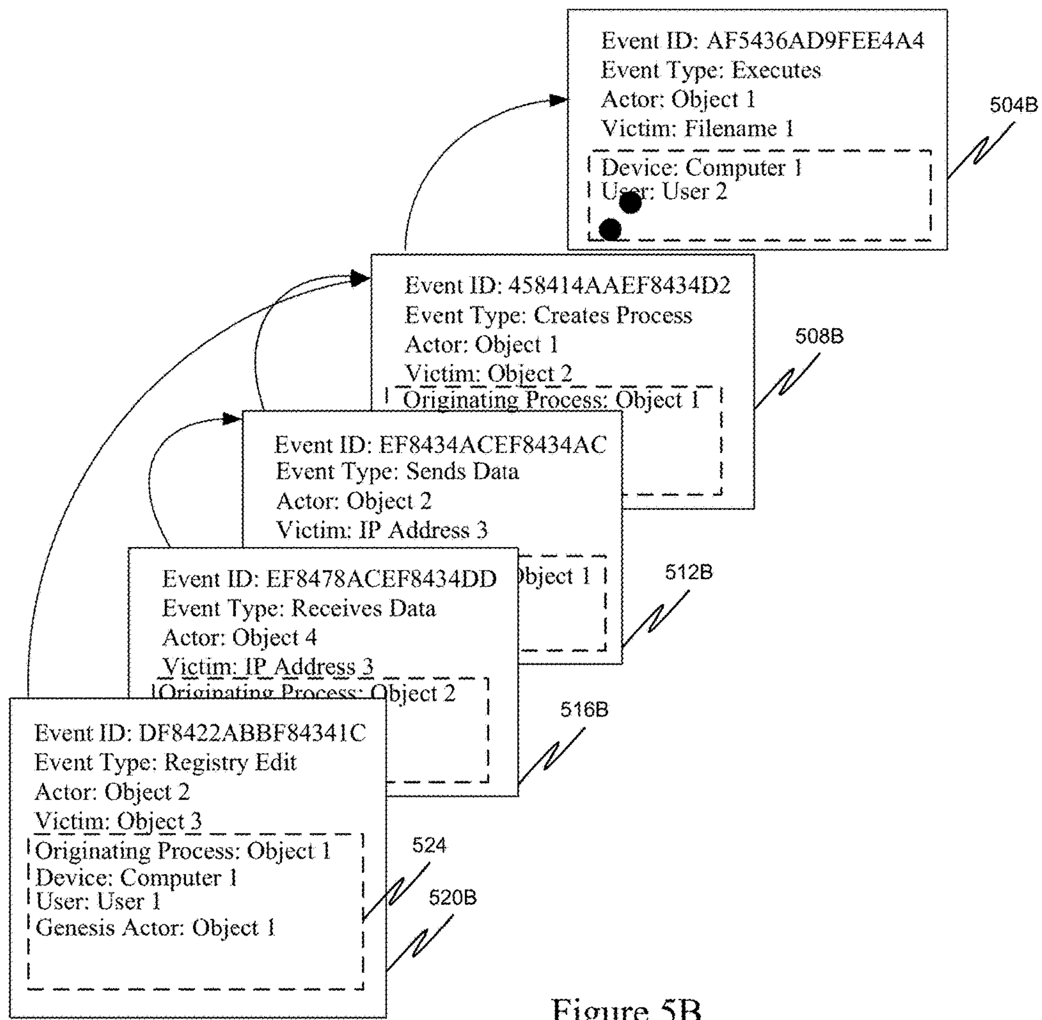

Event ID: AF5436AD9FEE4A4
Event Type: Executes
Actor: Object 1
Victim: Filename 1
504A
Event ID: 458414AAEF8434D2
Event Type: Creates Process
Actor: Object 1
Victim: Object 2
508A
Event ID: EF8434ACEF8434AC
Event Type: Sends Data
Actor: Object 2
Victim: IP Address 3
512A
Event ID: EF8478ACEF8434DD
Event Type: Receives Data
Actor: Object 4
Victim: IP Address 3
516A
Event ID: DF8422ABBF84341C
Event Type: Registry Edit
Actor: Object 2
Victim: Object 3
520A

Figure 5A



Event ID: AF5436AD9FEE4A4
Event Type: Executes
Actor: Object 1
Victim: Filename 1
Device: Computer 1
User: User 2
504B
Event ID: 458414AAEF8434D2
Event Type: Creates Process
Actor: Object 1
Victim: Object 2
Originating Process: Object 1
508B
Event ID: EF8434ACEF8434AC
Event Type: Sends Data
Actor: Object 2
Victim: IP Address 3
512B
Event ID: EF8478ACEF8434DD
Event Type: Receives Data
Actor: Object 4
Victim: IP Address 3
Originating Process: Object 2
516B
Event ID: DF8422ABBF84341C
Event Type: Registry Edit
Actor: Object 2
Victim: Object 3
Originating Process: Object 1
Device: Computer 1
User: User 1
Genesis Actor: Object 1
524
520B

Figure 5B

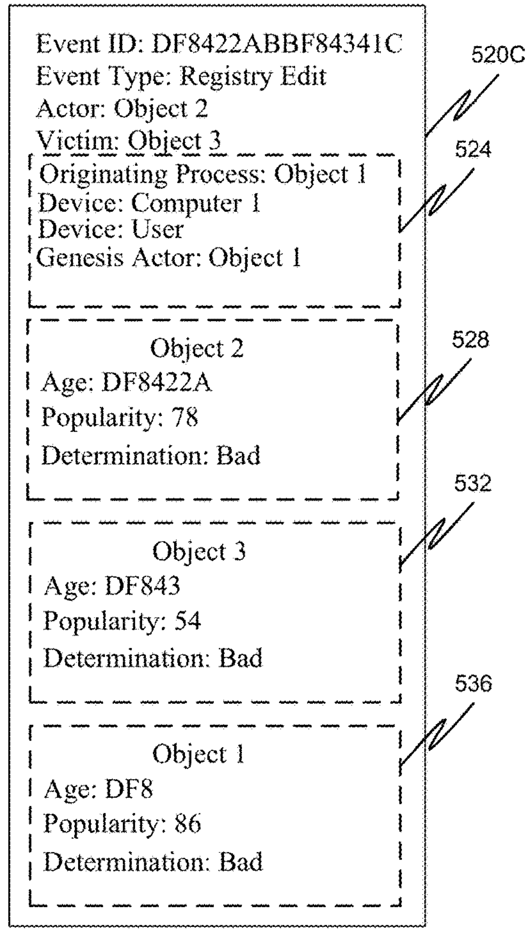

Event ID: DF8422ABBF84341C
Event Type: Registry Edit
Actor: Object 2
Victim: Object 3
Originating Process: Object 1
Device: Computer 1
Device: User
Genesis Actor: Object 1
520C
524
Object 2
Age: DF8422A
Popularity: 78
Determination: Bad
528
Object 3
Age: DF843
Popularity: 54
Determination: Bad
532
Object 1
Age: DF8
Popularity: 86
Determination: Bad
536

Figure 5C

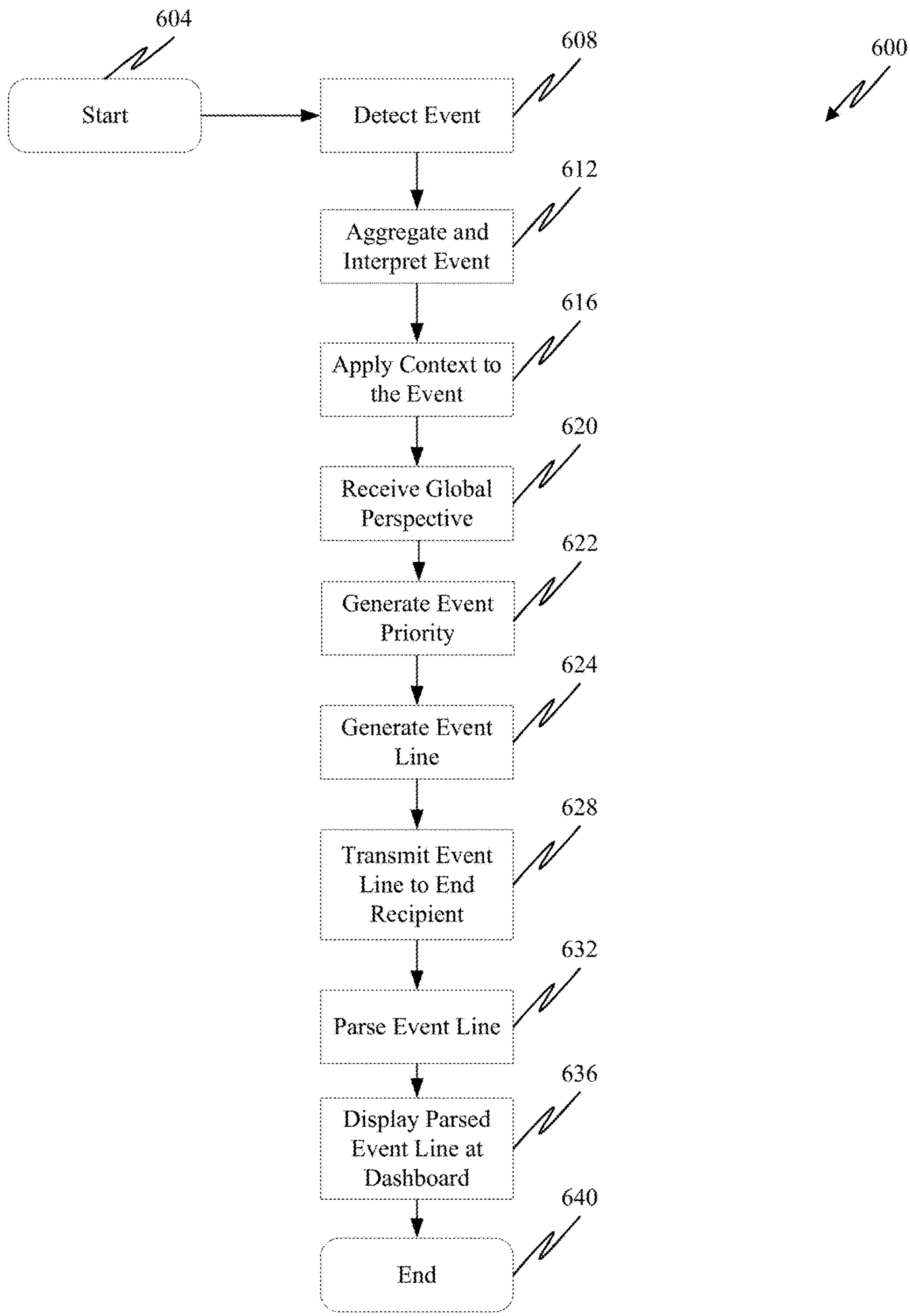
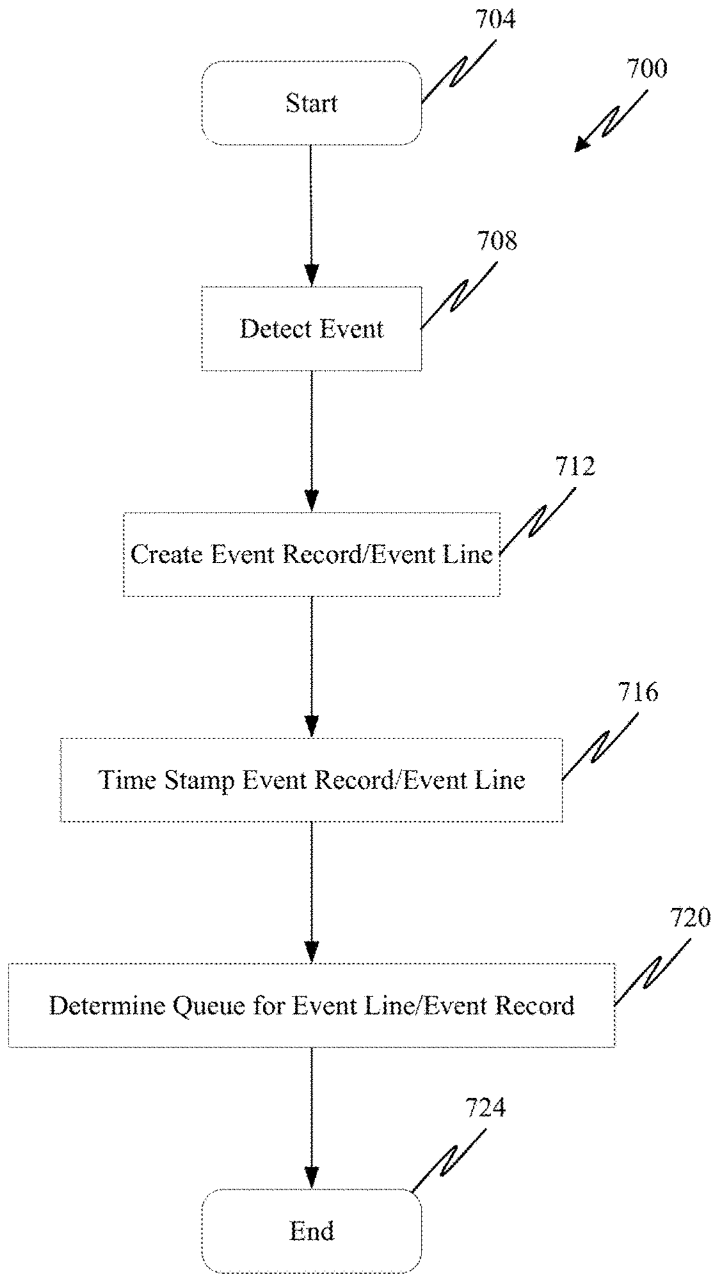

604
Start
608
Detect Event
612
Aggregate and Interpret Event
616
Apply Context to the Event
620
Receive Global Perspective
622
Generate Event Priority
624
Generate Event Line
628
Transmit Event Line to End Recipient
632
Parse Event Line
636
Display Parsed Event Line at Dashboard
640
End
600

Figure 6



700
704
Start
708
Detect Event
712
Create Event Record/Event Line
716
Time Stamp Event Record/Event Line
720
Determine Queue for Event Line/Event Record
724
End

Figure 7

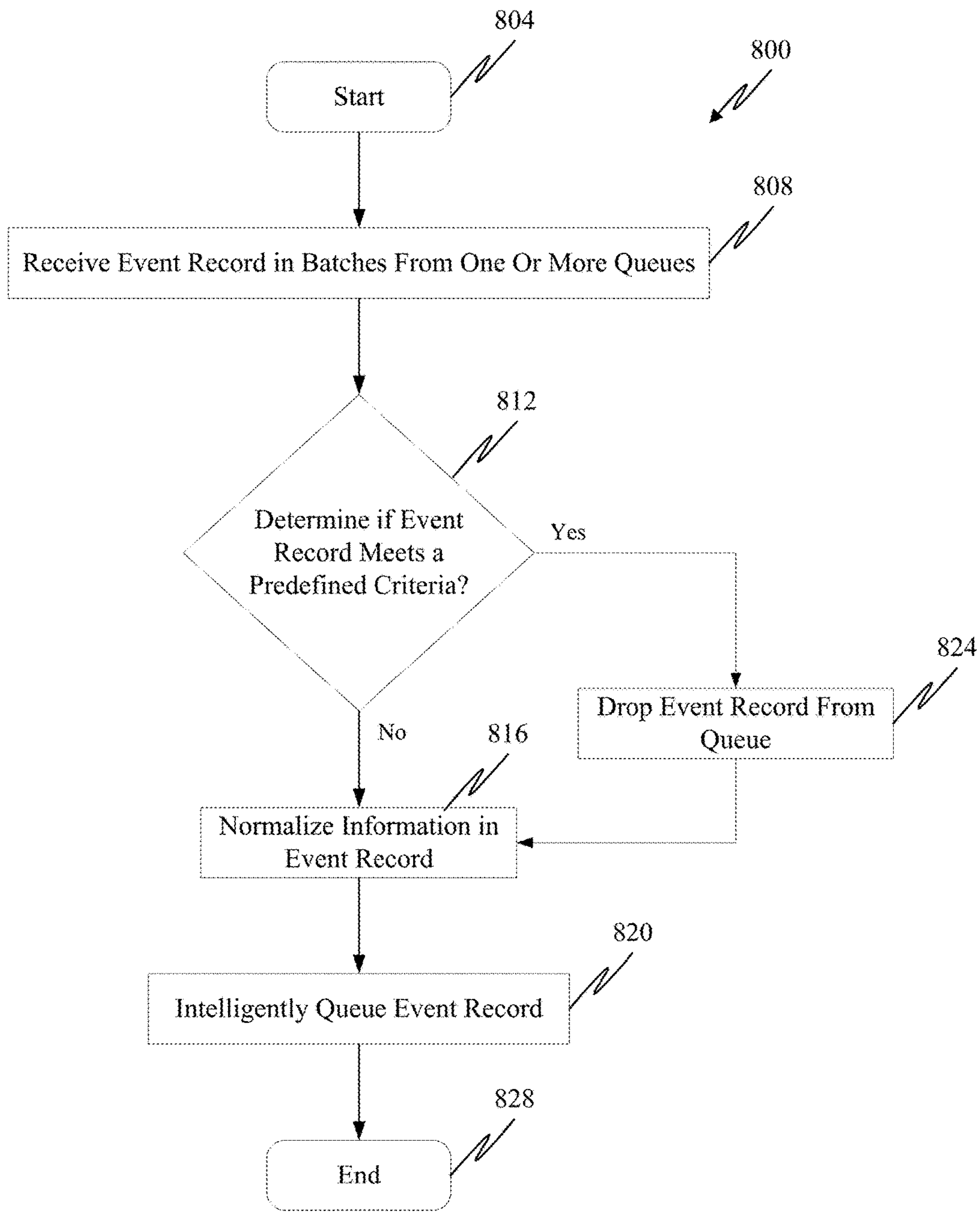

804
800
Start
808
Receive Event Record in Batches From One Or More Queues
812
Determine if Event Record Meets a Predefined Criteria?
Yes
824
Drop Event Record From Queue
No
816
Normalize Information in Event Record
820
Intelligently Queue Event Record
828
End

Figure 8

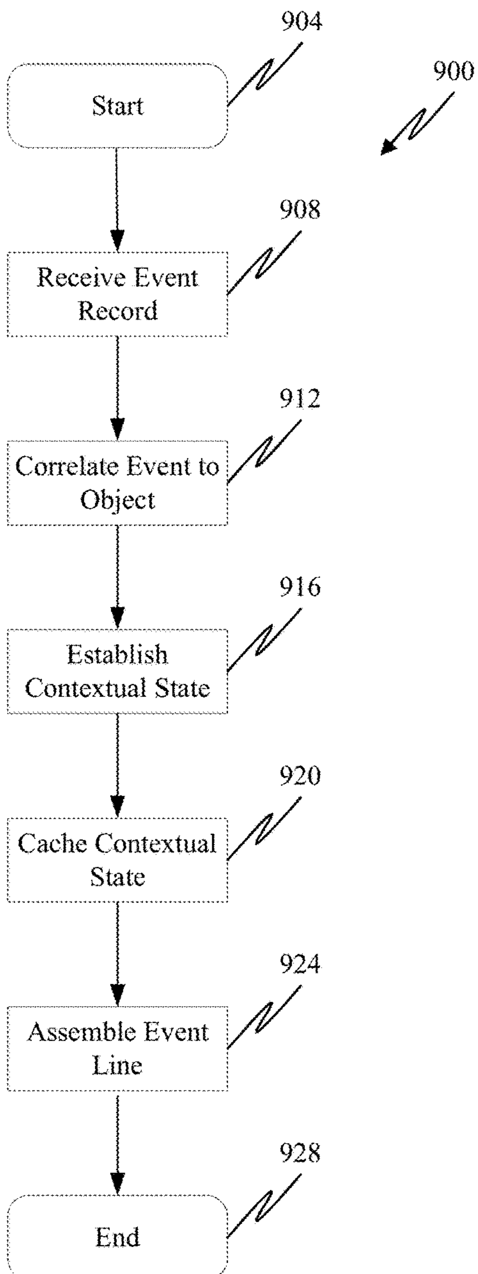

900
904
Start
908
Receive Event Record
912
Correlate Event to Object
916
Establish Contextual State
920
Cache Contextual State
924
Assemble Event Line
928
End

Figure 9

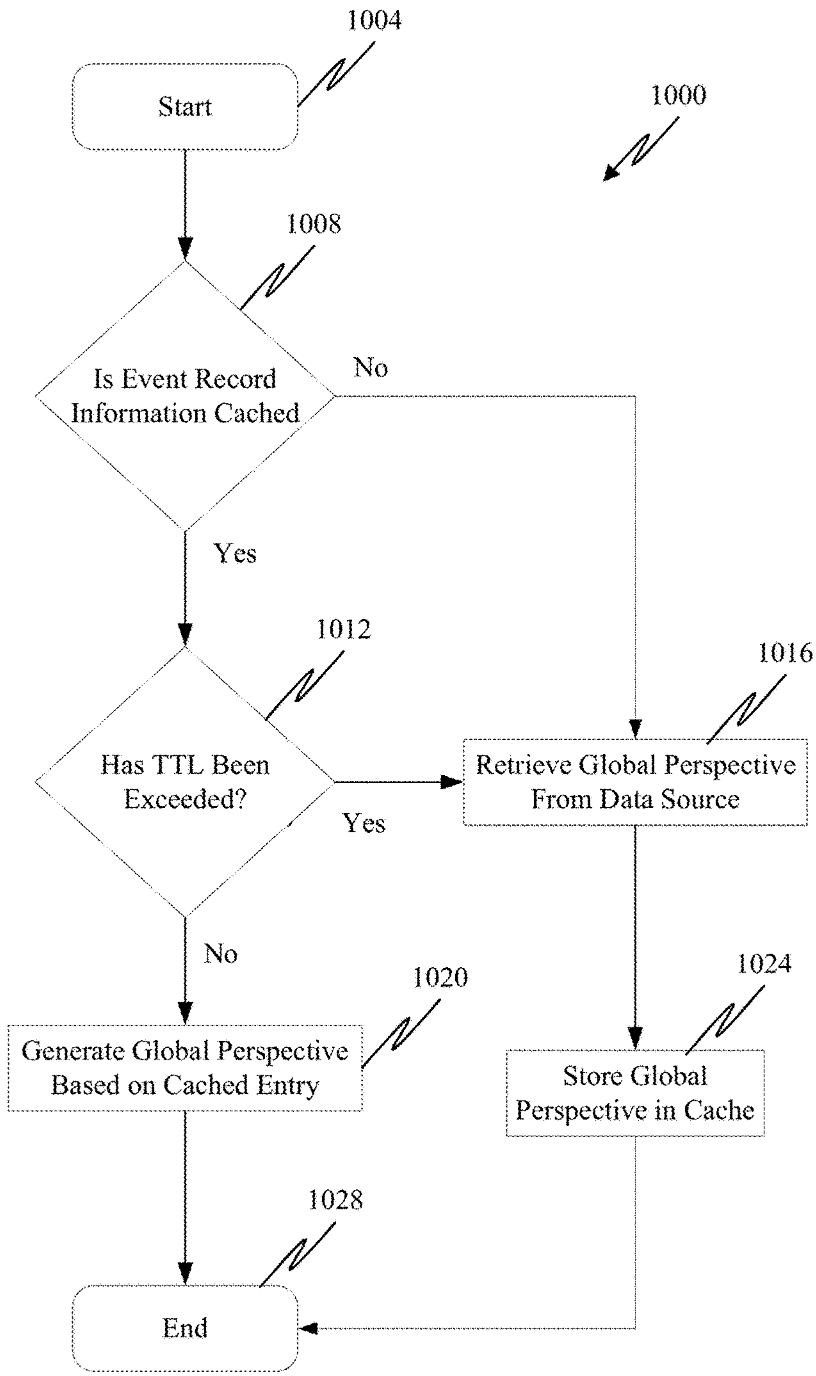

1004
Start
1000
1008
Is Event Record Information Cached
No
Yes
1012
1016
Has TTL Been Exceeded?
Retrieve Global Perspective From Data Source
Yes
No
1020
1024
Generate Global Perspective Based on Cached Entry
Store Global Perspective in Cache
1028
End

Figure 10

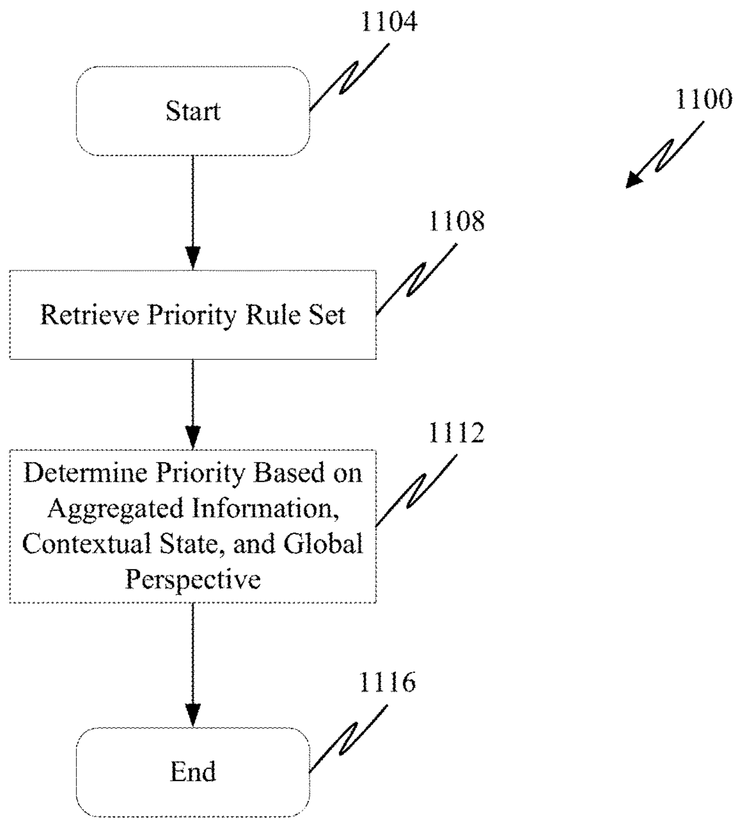

1104
Start
1100
1108
Retrieve Priority Rule Set
1112
Determine Priority Based on Aggregated Information, Contextual State, and Global Perspective
1116
End

Figure 11

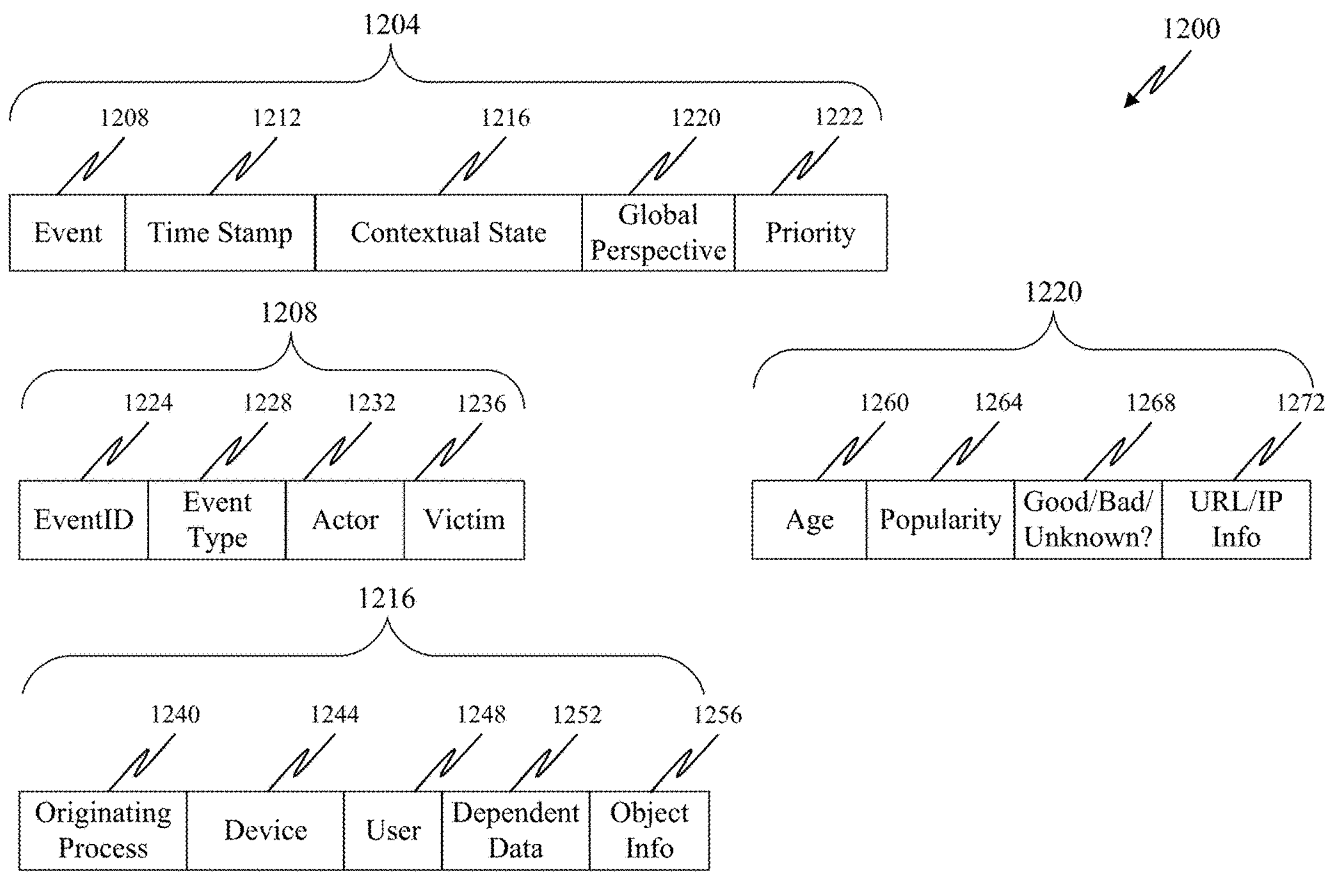

1200
1204
1208
1212
1216
1220
1222
Event
Time Stamp
Contextual State
Global Perspective
Priority
1208
1224
1228
1232
1236
EventID
Event Type
Actor
Victim
1220
1260
1264
1268
1272
Age
Popularity
Good/Bad/ Unknown?
URL/IP Info
1216
1240
1244
1248
1252
1256
Originating Process
Device
User
Dependent Data
Object Info

Figure 12

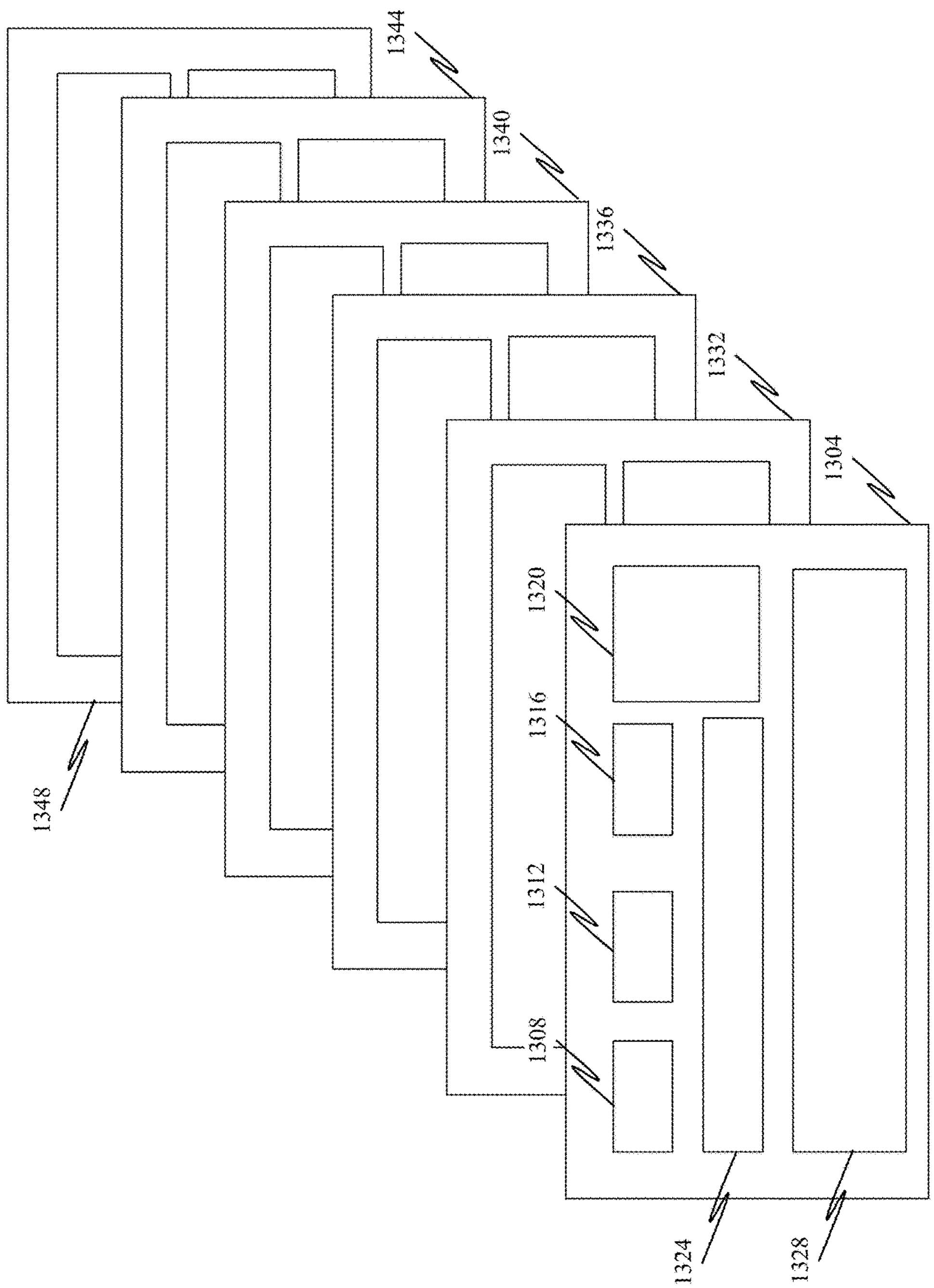


Figure 13

METHOD AND APPARATUS FOR PROVIDING FORENSIC VISIBILITY INTO SYSTEMS AND NETWORKS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Patent Application Ser. No. 61/819,470, filed May 3, 2013, the entire disclosure of which is hereby incorporated herein by reference.

FIELD

Methods and apparatuses for providing forensic visibility are described. More particularly, methods and systems for providing forensic visibility into client systems having full audit trails are described.

BACKGROUND

Network forensics generally relates to the capture, recording, and analysis of network events in order to discover the source of security attacks or other problem incidents. Network forensic systems generally utilize one of two approaches: the “catch it as you can approach” and the” stop, look, and listen approach.” “Catch it as you can” systems immediately write the packets to a disk file, buffering in memory as necessary, and perform analysis in batches. “Stop, look and listen” systems analyze the packets in memory, perform rudimentary data analysis and reduction, and write selected results to disk or to a database over the network.

However, existing network forensic tools, such as network sniffers and packet capturing tools, passively collect information about network traffic. Such tools may be able to detect and capture information associated with communication sessions, such as hostnames, ports, protocols, IP addresses, etc.; however, such tools tend to receive network events outside of a host or system. That is, a network forensic tool, such as a network sniffer, may reside at a network edge and may receive network events concerning the network from one or more hosts or systems. Accordingly, network edge devices have no ability to establish any sort of context from within the host or system that is generating the event. Therefore, devices residing at a network edge may monitor, record, and or analyze network traffic between the host and the device, but information that may be useful within the host is not obtained.

For example, FIG. **1** generally illustrates a comparative example including computing devices **104** and **108** in communication with a network edge device **116**. The network edge device **116** may be commonly referred to as a “sniffer” which may monitor, capture, and analyze network traffic between the network edge device **116** and the computing devices **104** and **108**. The network edge device **116** may further be in communication with a communication network **124**, such as the internet. Accordingly, the network edge device **116** may monitor, capture, and analyze network traffic between the computing devices **104** and **108** and between the computing devices **104/108** and the communication network **124**.

However, as previously mentioned, the network edge device **116** has no ability to monitor, capture, and/or analyze what is occurring within each of the computing device **104** and **108**. In at least one comparative example, computing device **108** may contain or otherwise be infected with one or more pieces of malware. The term “malware” is used herein to refer generally to any executable computer file or, more generally “object”, that is or contains malicious code, and thus includes viruses, Trojans, worms, spyware, adware, etc. and the like. Indeed, the network edge device **116** may be able to monitor, capture, and analyze communication traffic including the information, both encrypted and unencrypted, source and destination ports, source and destination IP address, and protocols utilized by the computing device **108**, but the network edge device cannot provide details regarding files modified, registry entries changed, and/or new processes that are created. Accordingly, the network edge device **116** has no way of knowing to what extent the virus may have impacted the computing device **108**, what additional files may have become infected, or what processes and/or objects operating are linked to the virus. What is needed therefore, is a solution that eliminates the guesswork and allows for end-to-end visibility of every event within a system and/or within a network.

SUMMARY

It is, therefore, one aspect of the present disclosure to provide a system and method whereby events occurring within a computing device are captured and additional context and a global perspective is provided for each capture event. For example, a sensor agent may provide visibility into occurrences across an environment, such as a networked environment, to ensure that an administrator is aware of any system changes and data communication in and out of computing devices residing on the network. In this implementation, six components, or modules, may be utilized in the event gathering and processing: low level system filters, a local aggregator and interpreter, a context analyzer, a global perspective module, an event priority module, and an event distributor may be utilized to provide forensic visibility into one or more computing devices of a communication network.

The system filters may be built upon the same or similar technology related to behavior monitoring and collection, as discussed in U.S. application Ser. No. 13/372,375 filed Feb. 13, 2012, “Methods and Apparatus for Dealing with Malware” (US-2012-0260340-A1), which is incorporated herein by reference in its entirety for all that it teaches and for all purposes. These filters intercept system events in a manner such that the operation of the system filter does not impact system performance, a key aspect of this solution. Events at this level tend to be very raw and generally not individually useful—on an average system, tens of millions of events take place every minute and the noise ratio can prevent forensic solutions from being able to provide sufficient value to the end consumer of their data due to the inability to quickly find important events. A product which impacts system performance will have considerably diminished value to an administrator and can negatively affect the results of an analysis undertaken. For example, a system filter that requires a majority of the available computing and system resources may render one or more communication devices unable to perform its primary function and may steer a real-time forensics analysis to an irrelevant analysis of events; irrelevant because such event analysis of information may be based on old and outdated event information. In some embodiments, events are gathered and timestamped accurately and sequentially, establishing a chronological order of system events for consumption at a higher level.

In at least one embodiment, the second component, the local aggregator and interpreter, receives events from the

low level system filters, collecting them in batches from the queues in which they are inserted in real-time, and transmits these events to a local service process where the events are held for further analysis. At this stage, intelligent caching may be invoked to drop events which are duplicates of recent events, and some data normalization may be performed to ensure consistency. Caching may be implemented sparingly but in critical areas to ensure redundant events are not continually reported—many applications perform thousands of the same events (network access to the same destination IP address, for example) and transmitting these events into an external log source would create a significant amount of unnecessary noise and potential performance degradation. The cache may integrate a "time to live" concept to prevent applications from hiding their traffic within their noise. The caches may automatically roll over to ensure sufficient space is always available without requiring exorbitant amounts of memory.

In addition, data normalization takes items such as kernel filenames, usermode filenames, process objects, registry objects, and other system structures and normalizes them into a standard format which is easy more easily consumed by automated parsers. Consolidating and normalizing data at this stage allows the data to be handled much more quickly at the higher levels to avoid reprocessing data and potentially missing events just because they were referred to slightly differently (\Device\HarddiskVolume1\Windows\file.exe instead of C:\Windows\file.exe for example). In some embodiments, an aspect of this subsystem provides the ability to process events asynchronously. The subsequent modules which will execute over the data can potentially take a considerable amount of time to complete but delaying the response for each event at this phase would result in a significant system slowdown. Instead, events are intelligently queued in multiple layers to minimize the number of individual threads but still ensure each event is given sufficient time to be independently processed. Because of this structure, there is no performance difference if an event takes one millisecond to be processed or one hour for the full series of routines to complete.

In at least some embodiments, the context analyzer is utilized to ensure that a user, such as a network administrator, is provided the deepest view of the most important data but still has access to the full range of data if they wish to see it. The analysis starts by correlating events to objects (their originating processes, devices, and users), and creating an audit trail for each individual event. This allows an administrator to forensically track back any event which occurs to what triggered it. As previously discussed, conventional products in this space receive events outside of the system (for example, as a network edge device) and therefore have no ability to establish context within the system. The present solution eliminates the guesswork and allows for end-to-end visibility of every event from the bottom of the stack to the top. As contextual state is established, it is cached and aggregated to be quickly queried in subsequent lookups further reducing latency and improving the responsiveness of the system.

After the contextual analysis completes, the sensor agent attempts to establish a global perspective for each event by calling the cloud database using an infrastructure similar to the infrastructure discussed in U.S. application Ser. No. 13/372,375 filed Feb. 13, 2012, "Methods and Apparatus for Dealing with Malware" (US-2012-0260340-A1), but with an additional layer capable of responding in a highly scalable manner which avoids the overhead generally associated with the complex rule processing within the overall database. This response includes information related to the age, popularity, and determination of an object—whether it is known good, bad, or unknown. This data may then be normalized and appended to each relevant portion of the event line. Further lookups are performed for network activity, assessing the reputation and category of each website and IP address as they are contacted.

The event priority module may contain priority logic, or one or more rule sets, to associate a priority to one or more events. The event priority module may utilize information from the local aggregator and interpreter, the context analysis module, and the global perspective module, to determine, or otherwise assign, a priority to an event; such a priority may be based on a rule set. The rule set may be provided by an administration system, backend system, end recipient **220**, or other component of a forensic system.

The event distributor may then batch and transmit the events to the end recipient after the global perspective has been applied. In some embodiments, the event distributor may batch and transmit the events to the end recipient after the even priority has been determined. In one implementation, a syslog protocol may be used to send each event as a line of log data to be processed by an end consumer—usually a display/query engine. The transmission of events requires an additional layer of asynchronous processing as the response time from the recipient cannot be allowed to slow down the higher level global perspective and contextual analysis work which is ongoing in other threads. Therefore, a transmission engine of the event distributor may build the final event line, packages it together with information about the host itself, and then send it directly into a separate queue where it will then be sent shortly after (usually a few seconds) while maintaining the pre-set timestamp of precisely when the event took place even if it is sent several seconds later.

The recipient of the event, whether hosted on a third party platform or using other proprietary software, may then display single events and/or perform wider analysis on groups of events, producing dashboard views and graphs to aid the administrator in identifying events that are most pertinent to their current investigation.

In accordance with at least one embodiment of the present disclosure, a method is provided, the method comprising gathering one or more events defining an action of a first object acting on a target; generating a contextual state for at least one of the one or more events by correlating the at least one event to an originating object, the contextual state including an indication of the originating object of the first object and an indication of at least one of a device on which the first object is executed and a user associated with the first object; obtaining a global perspective for the at least one event by obtaining information associated with one or more of the first object and the originating object, the information including at least one of age, popularity, a determination as to whether the first object is malware, a determination as to whether the originating object is malware; and Internet Protocol (IP) Address or Uniform Resource Locator (URL) information; assembling an event line including details associated with the at least one event, the details including information uniquely identifying each of the first object, the action of the first object, the target, and the originating object; and transmitting the assembled event line.

In accordance with further embodiments of the present disclosure, a system is provided, the system comprising a device including a communication interface; a processor; data storage; and a sensor agent stored on the data storage that is executable by the processor, wherein the sensor agent

is operable to: gather one or more events defining an action of a first object acting on a target; generate a contextual state for at least one of the one or more events by correlating the at least one event to an originating object, the contextual state including an indication of the originating object of the first object and an indication of at least one of a device on which the first object is executed and a user associated with the first object; obtain a global perspective for the at least one event by obtaining information associated with one or more of the first object and the originating object, the information including at least one of age, popularity, a determination as to whether the first object is malware, a determination as to whether the originating object is malware; and Internet Protocol (IP) Address or Uniform Resource Locator (URL) information; assemble an event line including details associated with the at least one event, the details including information uniquely identifying each of the first object, the action of the first object, the target, and the originating object; and transmit the assembled event line utilizing the communication interface.

In accordance with further embodiments of the present disclosure, a system is provided, the system comprising a device including a communication interface; a processor; data storage; and a sensor agent stored on the data storage that is executable by the processor, wherein the sensor agent is operable to: gather one or more events defining an action of a first object acting on a target; generate a contextual state for at least one of the one or more events by correlating the at least one event to an originating object, the contextual state including an indication of the originating object of the first object and an indication of at least one of a device on which the first object is executed and a user associated with the first object; assemble an event line including details associated with the at least one event, the details including information uniquely identifying each of the first object, the action of the first object, the target, and the originating object; and transmit the assembled event line utilizing the communication interface.

In accordance with at least one embodiment of the present disclosure, a method is provided, the method comprising obtaining an event line from one or more computing systems; parsing the event line to retrieve event information for at least one event, wherein the event information includes an event type, a contextual state for the event, a global perspective for the event, and a priority for the event; and displaying the event information for the at least one event to a display device.

In accordance with further embodiments of the present disclosure, a system is provided, the system comprising a device including a communication interface; a processor; data storage; a display device; and a parser stored on the data storage that is executable by the processor, wherein the parser is operable to: obtain an event line from one or more computing systems utilizing the communication interface; parse the event line to retrieve event information for at least one event, wherein the event information includes an event type, a contextual state for the event, a global perspective for the event, and a priority for the event; wherein the processor is operable to cause the event information for the at least one event to be displayed at a display device.

Additional features and advantages of embodiments of the present invention will become more readily apparent from the following description, particularly when taken together with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates a comparative example of a communication network, edge device, and one or more computing devices;

FIG. **2** depicts a communication network including one or more computing systems, global perspective information server, and one or more end recipients in accordance with an exemplary embodiment of the present disclosure;

FIG. **3** illustrates a block diagram depicting details of a computing system and a bridge in accordance with an exemplary embodiment of the present disclosure;

FIG. **4** illustrates a block diagram depicting details of a global perspective information server and an end recipient in accordance with an exemplary embodiment of the present disclosure;

FIGS. **5**A-C illustrates exemplary events, events having an applied contextual state, and events having an applied global perspective in accordance with an exemplary embodiment of the present disclosure;

FIG. **6** illustrates a flow chart depicting details of at least one embodiment in accordance with an exemplary embodiment of the present disclosure;

FIG. **7** illustrates a flow chart depicting operational details of a system filter in accordance with an exemplary embodiment of the present disclosure;

FIG. **8** illustrates a flow chart depicting operational details of a local aggregator and interpreter in accordance with an exemplary embodiment of the present disclosure;

FIG. **9** illustrates a flow chart depicting operational details of a context analyzer in accordance with an exemplary embodiment of the present disclosure;

FIG. **10** illustrates a flow chart depicting operational details of a global perspective unit and/or global perspective module in accordance with an exemplary embodiment of the present disclosure;

FIG. **11** illustrates a flow chart depicting operational details of an event priority generation unit in accordance with an exemplary embodiment of the present disclosure;

FIG. **12** depicts a first data structure employed in accordance with at least some embodiments of the present disclosure; and

FIG. **13** depicts one or more dashboards in accordance with at least some embodiments of the present disclosure.

DETAILED DESCRIPTION

FIG. **1** depicts aspects of a system **200** for providing forensic visibility into one or more computing devices **204**, **208**, and **212** in accordance with at least one embodiment of the present disclosure. In general, the system **200** includes one or more computing device **204**, each of which are interconnected to a global perspective information server **216** and end recipient **220** by one or more communication networks **212**. The computing device **204** may comprise a general purpose computer, such as but not limited to a laptop or desktop personal computer **204**B, a tablet computer, a smart phone **204**A, or other device capable of communications over a communication network **212** and capable of presenting content to an associated user. The computing device **204** may also comprise a processing unit that is interconnected to an affiliated input/output device.

The communication network **212** may include one or more networks capable of supporting communications between nodes of the system **200**, including but not limited to a computing devices **204**A-C, the global perspective information server **216** and the end recipient **220**. Examples of communication networks **212** include the Internet or any wide area network (WAN), local area network (LAN), or networks in various combinations. Other examples of communication networks **212** in accordance with embodiments of the present disclosure include wireless networks, cable

networks, satellite networks, and digital subscriber line (DSL) networks. In addition, different communication networks **212** may share a common physical infrastructure, for at least some portion of the physical network. For instance, a computing device **204** may be interconnected to a communication network **212** comprising the Internet and to a separate communication network between the computing device **204** and an end recipient **220**.

The computing device **204** includes and/or executes a sensor agent **208** in connection with the presentation of content to the user. In accordance with at least one embodiment of the present disclosure, six components, or modules, may be utilized by the sensor agent **208** to gather and process events. The sensor agent **208** may include, but is not limited to, low level system filters, a local aggregator and interpreter, a context analyzer, a global perspective module, an event priority module, and an event distributor. Of course, and as can be appreciated by those skilled in the art, all six modules may not be required by the sensor agent **208**. In at least one embodiment, the sensor agent may comprise fewer than six modules. That is, at least some functionality of the one or more modules may be provided by another device, such as a global perspective information server **216** for example. Moreover, it should be understood that the sensor agent **208** may include varying combination of modules. For example, the sensor agent **208** may include two of the six modules, three of the six modules, four of the six modules, or five of the six modules.

The system filters may intercept system events in a manner such that they do not impact system performance. Events may define those actions or behaviors of an object acting upon another object or some other entity. An object in this sense may be a computer file, part of a file or a sub-program, macro, web page or any other piece of code to be operated by or on the computer, or any other event whether executed, emulated, simulated or interpreted. Events may include, but are not limited to file activity, registry events, code events, network/internet events, process/thread events, window/GDI events, input events to name a few. An event may have three principal components: information and/or details describing the object performing the act (the "Actor"), information and/or details describing the act being performed (the "Event Type"), and information and/or details describing the object or identity of another entity upon which the act is being performed (the "Victim" or "Target"). For example, the event data might capture the identity, e.g. the IP address or URL, of a network entity with which an object is communicating, another program acting on the object or being acted on by the object, a database or IP registry entry being written to by the object, etc. While simple, this structure allows a limitless series of behaviors and relationships to be defined. Examples of the three components of an event might include, but are not limited to, those illustrated in Table 1.

TABLE 1

Actor	Event Type	Victim
Object 1	Creates Program	Object 2
Object 1	Sends data	IP Address 3
Object 1	Deletes Program	Object 4
Object 1	Executes	Object 2
Object 2	Creates registry key	Object 4

Moreover, in some embodiments, the Actor, Event Type, and/or Victim may be represented in the form of a so-called signature or key relating to the object key. Key generation is mainly to keep the data storage and transmission requirements as minimal as possible. The keys for each object may be formed by a hashing function operating on the object at the computing system **204**. Accordingly, an event may include the three principal components as a key of the object performing the act (the "Actor"), a key of the act being performed (the "Event Type"), and the key of the object or identity of another entity upon which the act is being performed (the "Victim" or "Target"). Additionally, events at this level tend to be very raw and generally not individually useful—on an average system, tens of millions of events take place every minute and the noise ratio can prevent forensic solutions from being able to provide sufficient value to the end consumer of their data due to the inability to quickly find important events. Accordingly, in some embodiments, events are gathered and time stamped accurately and sequentially, establishing a chronological order of system events for consumption at a higher level.

The second component, the local aggregator and interpreter, receives events from the low level system filters in batches from the queues in which they are inserted in real-time, and transmits them to the local service process where they are held for further analysis. In accordance with at least some embodiments of the present disclosure, the events may be cached and normalized such that kernel filenames, user mode filenames, process objects, registry objects, and other system structures are converted into a standard format for ease of consumption by one or more automated parsers. Following the aggregation and interpretation of the events, the context analyzer creates, or obtains, a contextual state by correlating events to objects and creating an audit trail for each individual event. In some instances, the sensor agent **208** of the computing device **204** may determine if contextual state information is available locally within a cache. If the contextual state information is not available locally, the sensor agent **208** of the computing device may obtain such contextual state information from a source of contextual state information accessible to the sensor agent **208**.

Contextual state information may include, but is not limited to, originating processes, devices, and users. Additionally, contextual state information may include GenesisActor information and ancillary data. GenesisActor information may contain the object that is not the direct Actor of an event but which is the ultimate parent of the event being performed. For example in the case of a software installation, the GenesisActor would be the object that the user or system first executed and that initiated the software installation process, e.g. Setup.exe. Ancillary data may include data associated with the event type, actor and victim. For example, an event such as that used to record the creation of a registry run key may identify the Actor object as creating a registry run key, the event type itself (e.g. "regrunkey"), and the Victim or subject of the registry run key. The ancillary data in this case would define the run key entry itself; the Hive, Key name and Value. As a contextual state is established for each event, the contextual state may be cached and aggregated to be quickly queried in subsequent lookups further reducing latency and improving the responsiveness of the computing device **204**. Contextual state information may be provided by the sensor agent **208** and/or a source of contextual state information accessible to the sensor agent **208**. For example, the global perspective information server **216**, in some embodiments, may be able to provide contextual state information to the sensor agent **208**

for one or more events. Moreover, such contextual state information may be provided from one or more other computing devices **204**.

After the contextual analysis completes, the sensor agent **208** establishes a global perspective for each event by calling on the global perspective information server **224**. For example, the sensor agent **208** of the computing device **204** may determine if global perspective information is available locally within a cache. If the global perspective information is not available locally, the sensor agent **208** of the computing device **204** may issue a global perspective request **224** utilizing the communication network **212** and destined for the global perspective information server **216**. The global perspective information server **216** may respond with global prospective information **228** and such information may be provided to the sensor agent **208** associated with the computing device **204**.

The global perspective information **228** may include information related to the age, popularity, and determination of an object of the event—whether it is known good, bad, or unknown. The global perspective information may be retrieved or otherwise obtained for each object of an event and any objects provided as a part of a contextual information state. In some instances, the reputation and category of each website and IP address are obtained as they are contacted. Such information may then be normalized.

The event priority module may contain priority logic, or one or more rule sets, to associate a priority to one or more events. The event priority module may utilize information from the local aggregator and interpreter, the context analysis module, and the global perspective module, to determine, or otherwise assign, a priority to an event; such a priority may be based on a rule set. The rule set may be provided by an administration system, backend system, end recipient **220**, or other component of a forensic system. For example, a rule set may determine that an event meeting certain conditions be assigned a priority according to a scale of severity. For instance, a rule set may specify that an event having a specific actor, victim, and/or origination process together with a global perspective indicating a specified age, popularity, malware determination, and/or IP/URL reputation be assigned an event priority representative of a potential risk assigned to the computing device **204** and/or network. Accordingly, when the event line containing event priority information is received at an end recipient **220**, the end recipient **220** may be able to more easily identify events, and/or instances of malware, that require more immediate attention.

An event distributor of the sensor agent **208** may then create an event line comprising information from the event, information from the contextual state, information from the global perspective, and information from regarding the priority of the event. Alternatively, or in addition, an event distributor of the global perspective information server **216** may create an event line comprising information from the event, information from the contextual state, and information from the global perspective.

Accordingly, once the event distributor of the sensor agent **208** and/or the event distributor of the global perspective information server **216** create the event line for at least one event, the event line **244**A-B may be sent to an end recipient **220** from the computing device **204** and/or the global perspective information server **216**. The event distributor may batch and transmit the events to the end recipient **220** after the global perspective has been applied. In at least one implementation, the syslog protocol is utilized to send each event as a line of log data to be processed by an end consume such as a display/query engine like the product "splunk".

Alternatively, or in addition, some embodiments of the present disclosure may feature an optional bridge **222** which may be a dedicated device residing locally on a network. The bridge **222** may aggregate communications to and from one or more computing system **204** such that information to and from sensor agent **208** is cached at the bridge **222**, optimizing global perspective requests and other communications. That is, the bridge may be shared across some or all devices in a network. In some embodiments, the bridge **222** may act as a local cache or data store for caching requests to the global perspective information server **216** and event lines directed to the end recipient **220**. Additionally, the bridge **222** may perform selective or intelligent caching of global perspective information such that subsequent lookups or requests of global perspectives may be served from the local cache of the bridge **222**.

FIG. **3** illustrates additional aspects of a system **200** in accordance with embodiments of the present invention, and in particular illustrates additional features and components of a computing device **204** and a bridge **222**. The computing device **204** may comprise a general purpose computer, tablet computer, smart phone, or other device or federated group of devices capable of supporting communications over a communication network **212**. The computing device **204** may include a processor **316**, memory **304**, one or more user input devices **308**, such as a keyboard and a pointing device, and one or more user output devices **312**, such as a display, speaker, and/or printer. Alternatively, or in addition, the user input **308** and the user output **312** may be combined into one device, such as a touch screen display. Computing device **204** may further include a communication interface **328** for communicating with another computing device **204**, a global perspective information server **216**, an end recipient **220**, and/or the communication network **212**.

Processor **316** is provided to execute instructions contained within memory **304** and/or storage **320**. As such, the functionality of the computing device is typically stored in memory **304** and/or storage **320** in the form of instructions and carried out by the processor **304** executing such instructions. Accordingly, the processor **316** may be implemented as any suitable type of microprocessor or similar type of processing chip. One example of the processor **304** may include any general-purpose programmable processor, digital signal processor (DSP) or controller for executing application programming contained within memory **304** and/or storage **320**. Alternatively, or in addition, the processor **316**, memory **304**, and/or storage **320**, may be replaced or augmented with an application specific integrated circuit (ASIC), a programmable logic device (PLD), or a field programmable gate array (FPGA).

The memory **304** generally comprises software routines facilitating, in operation, pre-determined functionality of the computing device **204**. The memory **304** may be implemented using various types of electronic memory generally including at least one array of non-volatile memory cells (e.g., Erasable Programmable Read Only Memory (EPROM) cells or FLASH memory cells, etc.). The memory **304** may also include at least one array of dynamic random access memory (DRAM) cells. The content of the DRAM cells may be pre-programmed and write-protected thereafter, whereas other portions of the memory may selectively be modified or erased. The memory **304** may be used for either permanent data storage and/or temporary data storage.

The data storage **320** may generally include storage for programs and data. For example, data storage **320** may

provide storage for a sensor agent **208** including the system filters **340**, the local aggregator and interpreter **344**, the context analyzer **348**, the global perspective module **352**, the event distributor **356**, and/or the general operating system and other programs and data **324**. Additionally, the data storage **320** may also include an event store **328**, such as a database, for storing and/or caching event information, context information, global perspective information, and/or event lines. One or more components of the computing device **204** may communicate with one another utilizing a bus **336**. Alternatively, or in addition, the sensor agent **208** may be provided separate and apart from data storage **320**. That is, the sensor agent **208**, including the system filters **340**, the local aggregator and interpreter **344**, the context analyzer **348**, the global perspective module **352**, the event priority generation unit **358**, and the event distributor **356** may be implemented using a processor, such as a microprocessor **304**. Accordingly, the microprocessor **304** may be specifically programmed to carry out one or more functions of the sensor agent **208**. For example, the processor **304** may execute instructions associated with the sensor agent **208**, including the system filters **340**, the local aggregator and interpreter **344**, the context analyzer **348**, the global perspective module **352**, the event priority generation unit **358**, and the event distributor contained within memory **304** and/or storage **320**. Alternatively, or in addition, the sensor agent **208**, including the system filters **340**, the local aggregator and interpreter **344**, the context analyzer **348**, the global perspective module **352**, the event priority generation unit **358**, and the event distributor **356** may share the processor **304**.

The system filters **340** intercept system events in a manner such that they do not impact system performance. In some embodiments, events are gathered and time stamped accurately and sequentially such that a chronological order of system events is maintained and may be provided for consumption at a higher level. As one example, the time stamp may correspond to the time at which the event took place. Such a timestamp may be added to the event and/or maintained at an event line. Moreover, the system events may then be sorted according to event type and/or processing resources such that the system events are inserted, or stored, in a queue in real-time.

The local aggregator and interpreter receives events from the system filters **340**, and collects the received events into batches from the queues in which they are inserted in real-time. The batches of events are then transmitted to a local service process where the events are held for further analysis. Such events may be held, or otherwise stored, within an event store **328**. At this stage, intelligent caching may be invoked to drop events which are duplicates of recent events, and some data normalization may be performed to ensure consistency. Caching may be implemented sparingly but in critical areas to ensure redundant events are not continually reported—many applications perform thousands of the same events (network access to the same destination IP address, for example) and transmitting these events into an external log source would create a significant amount of unnecessary noise and potential performance degradation. The cache may integrate a "time to live" concept to prevent applications from hiding their traffic within their noise. For example, events meeting predefined criteria may be dropped from the real-time system event queues. In some embodiments, if the predefined criteria include determining if a system event is the same or otherwise matches a previous system event, then the event may be dropped. Additionally, if a duplicate event occurs within a specified time, the event may be dropped from the queue. Alternatively, or in addition, if the duplicate system event occurs within a specified period of time, the duplicate system event may be modified to reflect that the duplicate system event is the same as a previously occurring system event. Accordingly, the later occurring duplicate system event may receive all contextual state information and global perspective information without requiring all of the determinations and/or lookups of such information. However, a time-to-live (TTL) functionality may be implemented on a cache basis to ensure that cached events, or previous events having context and/or a global perspective, are current. Accordingly, not all previous events having contextual information may be available to provide contextual information to a current event. As another option, the later occurring duplicate system event may be modified in a manner such that when the later occurring duplicate system event is interpreted by the end recipient **220**, the end recipient **220** may determine that the system event information may be the same as a previously parsed system event line. Accordingly, a full event line for the duplicate system event is not generated and/or transmitted saving bandwidth and information processing resources. Further, the caches may automatically roll over to ensure sufficient space is always available without requiring exorbitant amounts of memory.

The local aggregator and interpreter **344** may additionally perform data normalization on items such as kernel filenames, user mode filenames, process objects, registry objects, and other system structures and normalizes them into a standard format which is easy more easily consumed by automated parsers. Consolidating and normalizing data at this stage allows the data to be handled much more quickly at the higher levels to avoid reprocessing data and potentially missing events just because they were referred to slightly differently (\Device\HarddiskVolume1\Windows\file.exe instead of C:\Windows\file.exe for example). In some embodiments, an aspect of this subsystem provides the ability to process events asynchronously. The subsequent modules which will execute over the data may potentially take a considerable amount of time to complete but delaying the response for each event at this phase would result in a significant system slowdown. Instead, events are intelligently queued in multiple layers to minimize the number of individual threads but still ensure each event is given sufficient time to be independently processed. Because of this structure, there is little to no performance difference if an event takes one millisecond to be processed or one hour for the full series of routines to complete.

The context analyzer **348** ensures that a user at the end recipient **220** is provided with the deepest view of the most important data but still has access to the full range of data if they wish to see it. Accordingly, the context analyzer **348** correlates objects within system events to additional objects known to exist. For example, the context analyzer **348** may correlate an actor, victim, and/or event type to one or more originating processes, devices, and users. Such a correlation creates an audit trail for each individual event. As will be described later, this audit trail allows an administrator, at an end recipient **220** for example, to forensically track back any event which occurs to what triggered it. Accordingly, this additional information, including the originating processes, devices, and user, allows a contextual state to be established for each event. As each contextual state is established, the contextual state may be cached and aggregated to be quickly

queried in subsequent lookups further reducing latency and improving the performance of the system.

After the context analyzer completes, the sensor agent **208** may attempt to establish a global perspective for each event by calling the cloud database utilizing an infrastructure similar to the infrastructure discussed in U.S. application Ser. No. 13/372,375 filed Feb. 13, 2012, "Methods and Apparatus for Dealing with Malware" (US-2012-0260340-A1, but with an additional layer capable of responding in a highly scalable manner which avoids the overhead generally associated with the complex rule processing within the overall database. Such an additional layer may be named SkyMD5. The response from the SkyMD5 layer may include information related to the age, popularity, and determination of an object—whether it is known good, bad, or unknown. This data may then be normalized and appended to each relevant portion of the event line. Further lookups are performed for network activity, assessing the reputation and category of each website and IP address as they are contacted.

In addition, similar to the contextual analyzer, the global perspective unit may first determine whether or not global perspective information for the particular system event is available locally. For example, global perspective information may be cached or otherwise stored in the storage data **320**. In addition, a TTL approach may be additionally utilized to reduce the amount of traffic, or calls, to the global perspective information server **216**, by the global perspective unit **352** of the sensor agent **208**. Accordingly, contextual state information may only be available locally for a specified period of time and/or specified number of system events.

The event priority module **358** may contain priority logic, or one or more rule sets, to associate a priority to one or more events. The event priority module may utilize information from the local aggregator and interpreter **344**, the context analysis module **348**, and the global perspective module **352**, to determine, or otherwise assign, a priority to an event; such a priority may be based on a rule set. The rule set may be provided by an administration system, backend system, end recipient **220**, or other component of a forensic system. For example, a rule set may determine that an event meeting certain conditions be assigned a priority according to a scale of severity. Such an event priority may then be added to an existing event line or data structure.

The event distributor **356** may batch and transmit the events to the end recipient **220** after the global perspective has been applied. Alternatively, or in addition, the event distributor **356** may batch and transmit the events to the end recipient **220** after an event priority has been determined for the event. In one implementation, the syslog protocol is used to send each event as a line of log data to be processed by an end recipient **220**. The transmission of events requires an additional layer of asynchronous processing as the response time from the recipient cannot be allowed to slow down the higher level global perspective and contextual analysis work which is ongoing in other threads. Therefore, a transmission engine of the event distribute **356** builds the final event line, packages the event line together with information about the host itself, and then sends it directly into a separate queue where it will then be sent shortly after (usually a few seconds) while maintaining the pre-set timestamp of precisely when the event took place even if it is sent several seconds later. The separate queue may be a queue located in the storage data **320** and more specifically in the event store **328**.

The bridge **222** may comprise a general purpose computer and/or server or other device or federated group of devices capable of supporting communications over a communication network **212**. The bridge **222** may include a processor **382**, memory **362**, one or more user input devices **366**, such as a keyboard and a pointing device, and one or more user output devices **378**, such as a display, speaker, and/or printer. The bridge **222** may further include a communication interface **386** for communicating with another computing device **204**, an end recipient **220**, a global perspective information server **216**, and/or the communication network **212**. Processor **382** and memory **362** may be the same or similar as that which was described with respect to processor **316** and memory **304** respectively.

The data storage **370** may generally include storage for programs and data. For example, data storage **370** may provide storage for a cached event store **394** and/or the general operating system and other programs and data **374**. Optionally, as indicated by the dashed line, the cached event store **394** may be separate and apart from the data storage **370**. That is, the cached event store **394** functions independently and/or in cooperation with memory **362** and processor **382**.

The bridge **222** may be a dedicated device residing locally on a network. The bridge **222** may aggregate communications to and from one or more computing system **204** such that information to and from sensor agent **208** is cached and stored at the cached event store **394**, optimizing global perspective requests and other communications. Additionally, the bridge **222** may perform selective or intelligent caching of global perspective information such that subsequent lookups or requests of global perspectives may be served from the cached event data store **394**. One or more components of the bridge **222** may communicate with one another utilizing a bus **390**.

FIG. **4** illustrates additional aspects of a system **200** in accordance with embodiments of the present invention, and in particular illustrates additional features and components of a global perspective information server **216** and an end recipient **220**. The global perspective information server **216** may comprise a general purpose computer and/or server or other device or federated group of devices capable of supporting communications over a communication network **212**. The global perspective information server **216** may include a processor **464**, memory **452**, one or more user input devices **456**, such as a keyboard and a pointing device, and one or more user output devices **460**, such as a display, speaker, and/or printer. The global perspective information server **216** may further include a communication interface **468** for communicating with another computing device **204**, an end recipient **220**, and/or the communication network **212**. Processor **464** and memory **452** may be the same or similar as that which was described with respect to processor **316** and memory **304** respectively.

The data storage **488** may generally include storage for programs and data. For example, data storage **488** may provide storage for a global perspective module **476**, data store **486**, and/or the general operating system and other programs and data **484**. Optionally, each of the global perspective module **476**, the data store **486**, and/or the event distributor **492** may be separate and apart from the data storage **488**. That is, as indicated by the dashed line, each of the global perspective module **476**, the data store **486**, and/or the event distributor **492** functions independently and/or in cooperation with memory **452** and processor **464**.

The global perspective module **476** provides global perspective information at the request of a sensor agent **208**. For

example, and as previously described, the sensor agent **208** may transmit a global perspective request **224** to the global perspective information server **216** requesting global perspective information for one or more system events having a contextual state. Accordingly, the global perspective module **476** may provide the global prospective information **228** to the global perspective unit **352** of the computing system **204**. In addition, in some embodiments the event distributor functionality may reside at the global perspective information server **216**. Accordingly, in such an embodiment, the global perspective module **476** may receive event information having contextual state information. In such an instance, the global perspective module **476** may provide the global perspective information and store the system event, contextual information, and global perspective information in data store **486**. The event distributor **492** may then assemble the event line information, queue the event line for transmission, and transmit the event line to the end recipient **220**. The event distributor **492** may function and/or operate in a manner similar to or the same as the event distributor **356** of the sensor agent **208**. One or more components of the global perspective information server **216** may communicate with one another utilizing a bus **472**.

The end recipient **220** may comprise a general purpose computer and/or server or other device or federated group of devices capable of supporting communications over a communication network **212**. The end recipient **220** may include a processor **416**, memory **404**, one or more user input devices **408**, such as a keyboard and a pointing device, and one or more user output devices **412**, such as a display, speaker, and/or printer. The end recipient **220** may further include a communication interface **428** for communicating with another computing device **204**, a global perspective information server **216**, and/or the communication network **212**. Processor **416** and memory **404** may be the same or similar as that which was described with respect to processor **316** and memory **304** respectively.

The end recipient **220** may further include data storage **420** which may include storage for programs and data. For example, data storage **420** may provide storage for an event line parser **440**, a user interface generator **444**, a global information store **448**, and/or the general operating system and other programs and data **420**. Optionally, as indicated by the dashed line, each of the event line parser **440**, the user interface generator **444**, and the global information store **448** may be provided separate and apart from the data storage **436** where each of the event line parser **440**, a user interface generator **444**, and the global information store **448** functions independently and/or in cooperation with memory **404** and processor **416**. One or more components of the end recipient **220** may communicate with one another utilizing a bus **432**.

The event line parser **440** may parse an incoming event line **244**A-B and store the event lines, either parsed or whole, at the global information store **448**. Accordingly, the end recipient **220** may utilize the parsed event line to generate an user interface utilizing the user interface generator **444**. The generated user interface may then be displayed to a user at an user output device **412**. The user interface generator **444** may create one or more dashboards that provide information to a user of the end recipient **220**. Accordingly, dashboards may display correlated events collected by a sensor agent **204**. Such correlated events may comprise File System/Disk activity, network activity, and registry activity on the endpoint along with the global community data and IP and domain reputation data to provide a complete point of view of the computing system

204 activity and further provide a full meaning to every single reported event. Accordingly, spotting and highlighting behaviours in the event data from computing systems **204** may potentially assist in finding and determining whether such activity is malicious or not.

Moreover, there may exist multiple dashboards each of which provide a specific heuristic that targets a specific potential malicious behaviour. For example, a first dashboard generally looks for executables that have been found and executed from specific critical paths in the computing system **204**; those paths that are known to host malware executables (browser's cache, temporary files folder, system32 folder, auto start folder and so on). The dashboard may filter those files determined as good, unknown and bad from global community information. For unknown files, the dashboard may show their popularity at the global community, i.e. on how many computing systems **204** they have been seen—and if they establish outbound internet connections. If such files connect to the Internet, the dashboard may trace down and identify the specific connection and geolocate the destination address on a world map.

In another embodiment, a separate dashboard may analyze outbound connections from the analyzed endpoints. Accordingly, this dashboard may filter out and highlight connections to high-risk countries, i.e. those countries known to host malicious servers, and retrieve software that actually established the suspicious connections. The dashboard may also provide another point of view; that is, the dashboard may highlight all connections to specific IPs and domains that have been set up recently (e.g. in the last 60 days). All connections may be shown on a map view and the contacted IPs and domains are crosschecked with a database to determine their reputation and website category (if known).

As another example, a dashboard may highlight potential exploits detected on the endpoint by monitoring the most common software usually installed on an average computing system **204** and targeted by exploit attacks. The dashboard may generally identify software like Java, Adobe Acrobat, Microsoft Office, Flash Player and monitor if such software is dropping executable files on the system. Accordingly, if the software behaves in this manner, such a behaviour should be considered a suspicious behaviour, as software should not be dropping any executable files on the system. If such an occurrence is detected, the dashboard may analyze the dropped executable for its popularity in the global community and then identify potential outbound connections established by the just-dropped executable. If an outbound connection is found, the dashboard retrieves the destination IP and renders, or draws, the location of the IP address on a map view, such as a world map.

As another example, a separate dashboard may highlight those new processes created on the monitored computing systems **204** that are undetermined and have only been recently seen in the global community. For every process, a lifetime scheme is assembled which allows the administrator to understand when the undetermined file firstly landed on the endpoint and for how long time it has been active on the system. This dashboard may further provide specific events triggered by the suspicious process while active on the system.

As another example, a dashboard may highlight system processes that have been found to establish concurrent connections to different IPs in less than a specific timeframe and/or threshold, such as one second. If found, the system processes may be suspect and deemed to host malicious code; such malicious code having been injected into the

website. For every established connection, the dashboard may trace and identify the contacted country and the specific IP/domain. As another example, a dashboard may be generated by the user interface generator **448** such the dashboard monitors activities on remote storage services like Dropbox, showing the filename and size of the files being saved on a remote server. The dashboard may further monitor and identify sensitive or classified company data that may be in the process of being stolen or placed (i.e. saved, stored, etc.) on a non-approved storage location.

FIGS. **5**A-**5**C illustrate additional aspects of a system **200** in accordance with embodiments of the present invention, and in particular illustrates event capturing, the application of contextual state, and the application of a global perspective to such captured events. For example, events **508**A-**520**A are representative events that may have been captured by one or more system filters **340** of a sensor agent **204**. The system events **504**A-**520**A may include an Event ID, an Event Type, an Actor and a Victim. Although not illustrated, a timestamp may also be included in the event. As previously described, the actor and the victim may comprise objects. For example, in the system event **520**A, an event type of Registry Edit is captured where Object **2** is modifying Object **3**; accordingly, Object **3** may be the registry hive, a specific registry entry, and/or a specified registry value. Moreover each of the Event ID, Event Type, Actor, and Victim may be hashed resulting in a key that uniquely identifies the data.

As illustrated in FIG. **5**B, a contextual analysis may be applied to the captured events **504**B-**520**B. Accordingly, a context **520**B comprising an origination proceess, Device, User, and in some instances, a GenesisActor may further supplement or enhance the Event ID, Event Type, Actor, and Victim. Although illustrated as including an originating process, GenesisActor, device, and user, the contextual information forming the contextual state **520**B may include more or less information and in varying combinations. Moreover, as illustrated in FIG. **5**B, the originating process or object may determined based on a previous event that initiated or otherwise created the current process and/or object. Additionally, as illustrated in FIG. **5**B, the GenesisActor is determined based on a chain of previous events. For example, the Actor of Object **1** for system event **504**B of Event Type "executes" operates on Filename **1**. Object **1** subsequently creates a process Object **2** in **508**B which is further utilized in both **516**B and **520**B. Accordingly, as the Object **1** initiated subsequent events, Object **1** may be determined to be the GenesisActor for **520**B, **516**B, **512**B, and **508**B.

As illustrated in FIG. **5**C, a global perspective is applied to event **520**C. The global perspective may comprise information for each object in the event and/or the contextual state **524**. For example, a global perspective may be obtained for Object **2**, Object **3**, and Object **1** as illustrated. Accordingly, a global perspective **528** indicating the age of Object **2**, the popularity of Object **2**, the determination of Object **2**, and/or additional IP/URL information of Object **2** may be provided. Similar global perspective information **532**, **536** may be obtained for Object **3** and Object **1** respectfully. Accordingly, an event line may be assembled utilizing the event information, contextual state, and global perspective information. In addition, the global perspective may provide additional information pertaining to an IP/URL address. For example, such information may include a categoryID of the IP/URL, a confidence of such a categoryID, a reputation, geolocation information such as which country the IP/URL is associated with, and/or a threat history associated with the IP/URL.

Referring now to FIG. **6**, a method **600** for providing forensic visibility into a system is provided in accordance with embodiments of the present disclosure. Method **600** is in embodiments, performed by a device, such as the sensor agent **204** and/or the end recipient **220**. More specifically, one or more hardware and software components may be involved in performing method **600**. In one embodiment, one or more of the previously described units perform one or more of the steps of method **600**. The method **600** may be executed as a set of computer-executable instructions executed by a computer system and encoded or stored on a computer-readable medium. Hereinafter, the method **600** shall be explained with reference to systems, components, units, software, etc. described with FIGS. **1**-**5**C.

Method **600** may continuously flow in a loop, flow according to a timed event, or flow according to a change in an operating or status parameter. Method **600** is initiated at step **604** where events of a computing system **204** may be initiated. At step **608**, such an event may be detected. As previously discussed, the events, which may comprise system events, may be detected by one or more system filters **340** of a sensor agent **208**. The sensor agent **208** may be comparatively small compared with other commercially available forensic tools and anti-malware software. This may be achieved by the sensor agent **204** being developed in a low-level language, having direct access to system resources such as the video display, disk, memory, the network(s), and without the incorporation of many standard code or dynamic linked libraries to perform these functions. Memory usage may be optimized by storing data in a local database structure that provides the ability to refer to objects by a unique identifier rather than requiring full filenames or signatures. All unnecessary dynamic link libraries are unloaded from the process immediately as they are identified as no longer being used, and background threads are merged to reduce CPU usage. A small, efficient sensor agent **204** may be deployed more quickly and may be used alongside other programs, including other security programs, with less load on or impact on the computer's performance. This approach also has the advantage of having less surface area for attack by malware making it inherently more secure.

Method **600** then proceeds to step **612** where the local aggregator and interpreter **344** receives events from the low level system filters **340**, collecting them in batches from the queues in which they are inserted in real-time. As previously discussed, intelligent caching may be invoked to drop events which are duplicates of recent events. Further, some data normalization may be performed at step **612** to ensure consistency. Further still, caching may be implemented sparingly but in critical areas to ensure redundant events are not continually reported. Method **600** then proceeds to step **616** where a contextual state may be applied by a context analyzer **348** to the event information. Accordingly, the context analyzer correlates events to objects such that an audit trail for each individual event is created. As a contextual state is established, it is cached and aggregated to be quickly queried in subsequent lookups further reducing latency and improving the responsiveness of the system. Once a contextual state is applied, method **600** then proceeds to step **620** where a global perspective is applied for each event and/or each object of the event and contextual state. Accordingly, information related to the age, popularity, and determination of an object—whether it is known good, bad, or unknown is obtained. In addition, as previously mentioned, the global perspective may contain information

pertaining specifically to an IP/URL address. For example, the global perspective may include information relating network activity and an assessment of the reputation and category of each website and IP address as they are contacted. This data may then be normalized. Method **600** then proceeds to step **622** where an event priority may be applied to the event. As previously discussed, the event priority may be obtained utilizing a rule set and applying such a rule set to aggregated event information, contextual state information, and global perspective information. Accordingly, method **600** then proceeds to step **624** where such data is appended to each relevant portion of an event line.

Method **600** may then proceed to step **628** where the event distributor **356** may transmit the event line **244** to the end recipient **220**. The event distributor may first assemble the information and transmit the batched information after the global perspective has been applied. Once at the end recipient, the method **600** may then parse the event line at step **632** to filter out one or more events. Accordingly, the events may be stored in the data store **480**. Method **600** may then display the parsed event line at one or more dashboards in accordance with step **636**. Method **600** then ends at step **640**.

Referring now to FIG. **7**, additional details regarding step **608** of method **600** is provided in method **700**. Method **700** is in embodiments, performed by a device, such as the sensor agent **204** and/or the system filter **340**. More specifically, one or more hardware and software components may be involved in performing method **700**. In one embodiment, one or more of the previously described units perform one or more of the steps of method **700**. The method **700** may be executed as a set of computer-executable instructions executed by a computer system and encoded or stored on a computer-readable medium. Hereinafter, the method **700** shall be explained with reference to systems, components, units, software, etc. described with FIGS. **1-6**.

Method **700** may continuously flow in a loop, flow according to a timed event, or flow according to a change in an operating or status parameter. Method **700** is initiated at step **704** where events of a computing system **204** may be initiated. At step **708**, such an event may be detected. As previously discussed, the events, which may comprise system events, may be detected by one or more system filters **340** of a sensor agent **208**. Accordingly, at step **712**, an event record may be created. The event record may be the same or similar to that which was described with respect to FIG. **5**A. Accordingly, the event record may include an Event ID, and Event Type, an Actor, and a Victim. Moreover, in some embodiments, instead of creating a separate event record and a separate event line, a single event line for each event may be utilized. That is, the event record may be the event line. Accordingly, at step **716** the event line and/or the event record may be time stamped and stored. At step **720**, method **700** may then determine which queue the event record or event line, should be assigned. In some embodiments, the event record/event line may be assigned to a specified queue based on the event type. Alternatively, or in addition, the event record/event line may be assigned to a specified queue based on the needed resources and the available resources. Method **700** then ends at step **724**.

Referring now to FIG. **8**, additional details regarding step **612** of method **600** is provided in method **800**. Method **800** is in embodiments, performed by a device, such as the sensor agent **204** and/or the local aggregator and interpreter **344**. More specifically, one or more hardware and software components may be involved in performing method **800**. In one embodiment, one or more of the previously described units perform one or more of the steps of method **800**. The method **800** may be executed as a set of computer-executable instructions executed by a computer system and encoded or stored on a computer-readable medium. Hereinafter, the method **800** shall be explained with reference to systems, components, units, software, etc. described with FIGS. **1-7**.

Method **800** may continuously flow in a loop, flow according to a timed event, or flow according to a change in an operating or status parameter. Method **800** is initiated at step **804**. Method **800** then proceeds to step **808** where batches of event records/event lines are received form one or more previously described queues. For each event, method **800** proceeds to determine if the event record/event line meets a predefined criteria. For example, if the current event record/event line matches or is otherwise the same as a previous event record/event line, method **800** may proceed to step **824** where the current event record/event line may be dropped. Alternatively, or in addition, the dropped event record/event line may be modified such that either the contextual analyzer **348** and/or the global perspective unit **352** may be able to determine a contextual state and global perspective based on a previously captured event record/event line. Alternatively, or in addition, the dropped event record/event line may be modified to refer to a previously captured event record/event line such that upon parsing by the end recipient **220**, the current event line/event record is provided meaning from a previously parsed event record/event line

Accordingly, method **800** may then proceed to step **816** where the information may be normalized. The normalization may be implemented with a database which takes complete path names and translates them to representatives paths. So, for example, the Windows\file.exe would end up normalized to % windir %\file.exe, allowing rules to be created for any full path. Similar logic is utilized for paths like % appdata % which would have unique elements per-user and would be difficult to manage if not normalized. Accordingly, once the information in the event record/event line is normalized, method **800** may proceed to step **820** where the events may be intelligently queued. For instance, each event record/event line may be placed in a queue according to an event type. Method **800** may then end at step **828**.

Referring now to FIG. **9**, additional details regarding step **616** of method **600** is provided in method **900**. Method **900** is in embodiments, performed by a device, such as the sensor agent **204** and/or the context analyzer **348**. More specifically, one or more hardware and software components may be involved in performing method **900**. In one embodiment, one or more of the previously described units perform one or more of the steps of method **900**. The method **900** may be executed as a set of computer-executable instructions executed by a computer system and encoded or stored on a computer-readable medium. Hereinafter, the method **900** shall be explained with reference to systems, components, units, software, etc. described with FIGS. **1-8**.

Method **900** may continuously flow in a loop, flow according to a timed event, or flow according to a change in an operating or status parameter. Method **900** is initiated at step **904**. Method **900** then proceeds to step **908** where event records/event lines are received. Method **900** then proceeds to step **912** where objects of the event record/event line are correlated to objects in a contextual state. For example, and as illustrated in at least FIG. **5**B, a contextual state may include originating processes, devices, and users. Accordingly, once objects of the event record/event line have been correlated, method **900** proceeds to step **916** where the event

record/event line includes an established contextual state. In some embodiments consistent with the present disclosure, the contextual state may already be cached and may be obtained based on the event record/event line information. As one example, based on the event type, actor, and victim, it may be possible to recover a contextual state that includes an originating process, the device, and the user. However, not all cached instances of cached contextual state will suffice as different users may utilize a computing device **204**.

Method **900** then proceeds to step **920** where the contextual state may be cached and made accessible for later access. In accordance with at least one embodiment, the event line may be partially assembled where the partially assembled event line may include various combinations of the event type, the actor object, the victim object, and the origination process. Accordingly, the partially assembled event line may later be passed to a global perspective information server. As such, method **900** ends at step **928**.

Referring now to FIG. **10**, additional details regarding step **620** of method **600** is provided in method **1000**. Method **1000** is in embodiments, performed by a device, such as the sensor agent **204** and/or the global perspective unit **352**, the global perspective module **476**, and/or the global perspective information server **216**. More specifically, one or more hardware and software components may be involved in performing method **1000**. In one embodiment, one or more of the previously described units perform one or more of the steps of method **1000**. The method **1000** may be executed as a set of computer-executable instructions executed by a computer system and encoded or stored on a computer-readable medium. Hereinafter, the method **1000** shall be explained with reference to systems, components, units, software, etc. described with FIGS. **1-9**.

Method **1000** may continuously flow in a loop, flow according to a timed event, or flow according to a change in an operating or status parameter. Method **1000** is initiated at step **1004**. Method **1000** then proceeds to step **1008** where method **1000** determines whether event record/event line information is cached. Such a determination may be based upon whether the event information, such as the event type, the actor, and/or the victim, match a predetermined criteria. For instance, not all event types may be cached. In one embodiment, global perspective information caching will cache a specific number of event types occurring within a specified time period. In other instances, all global perspective information for all events may be cached. Accordingly, if it is determined that global perspective information is not cached, method **1000** proceeds to step **1016** where the global perspective information server may be contacted. Accordingly, such a request may provide the global perspective information server **216** with information in an event record and/or a partially assembled event line. Accordingly, the global perspective information or a particular event may be retrieved. The retrieved global perspective information may then be stored at step **1024**. For example, such information may be stored in event store **328**.

Otherwise, if global perspective information is cached for an event record/event line, the method **1000** may proceed to step **1012** were method **1000** may determine whether a time-to-live (TTL) has been exceeded. In some instances, the cached information may expire based on the frequency of occurrence for a particular event. For example, if an event occurs with a high frequency, the global perspective unit **352** may determine that for efficiency reasons, either resources and/or bandwidth, caching for a particular event would be beneficial to improving the overall efficiency of the sensor agent **208** and/or computing device **204**. Accordingly, the method **1000** then proceeds to step **1020** where the global perspective for the event is retrieved based on the cached entry. Otherwise, if the occurrence of the same event happens at a reduced frequency such that it would not be beneficial to cache global perspective information, it might be more beneficial to directly lookup and retrieve such information directly from the global perspective information server **216**. Accordingly, the method **1000** proceeds to step **1016** where global perspective information is received from the global perspective information server. In some instances, the cache of global perspective events may expire according to a predetermined time. At step **1024**, method **1000** may proceeds to step **1028** where method **1000** ends.

Referring now to FIG. **11**, additional details regarding step **624** of method **600** is provided in method **1100**. Method **1100** is in embodiments, performed by a device, such as the sensor agent **204**. More specifically, one or more hardware and software components may be involved in performing method **1100**. In one embodiment, one or more of the previously described units perform one or more of the steps of method **1100**. The method **1100** may be executed as a set of computer-executable instructions executed by a computer system and encoded or stored on a computer-readable medium. Hereinafter, the method **1100** shall be explained with reference to systems, components, units, software, etc. described with FIGS. **1-10**.

Method **1100** may continuously flow in a loop, flow according to a timed event, or flow according to a change in an operating or status parameter. Method **1100** is initiated at step **1104**. Method **1100** then proceeds to step **1108** where a priority rule set is retrieved. The priority rule set may be retrieved directly from an administration system and/or may be received locally from a storage location associated with the sensor agent **208** and/or the data storage **320**. The priority rule set may have one or more predefined criteria for assigning a priority level to an event. The priority level may represent a perceived severity or risk in which such an event poses. Method **1100** then proceeds to step **1112** where an event priority, or level, is assigned to the event.

In at least one embodiment, the priority level may be derived from aggregated and interpreted event information from the local aggregator and interpreter **344**, context analyzer **348**, and global perspective unit **352**. Accordingly, an event having a specified Actor, Victim, Target, originating process, or the like together with a global perspective indicating that at least one of the Actor, Victim, Target, origination process has been determined to be malware may result in a rule set that assigns a priority level to the event indicating that such the event should be immediately brought to the attention of an administrator. Accordingly, the event may be said to have a "high" priority or be a "high" priority event. Method **1100** may then end at step **1116**. With reference to FIG. **12**, details of a data structure **1200** will be described in accordance with embodiments of the present disclosure. In some embodiments, the data structure **1200** may be used as an event line and may contain details specific to an event, context, and global perspective. For example, an event line **1204** may generally include, without limitation, event information **1208**, a time stamp **1212**, context state information **126**, global perspective information **1220**, and a priority level **1222**, where the priority may be a value that ranges between a low and a high number. In some embodiments, the event field **1208** may further include Event ID information **1224**, Event Type information **1228**, Actor **1232**, and Victim information **1236**. In some embodiments, the context state **1216** may include, without limitation, Originating Process **1240** information, Device information

1244, User information **1248**, dependent data such as, but not limited to additional meta data and/or ancillary data describing or containing detailed information about one or more objects **1252**, and object info **1256**. In addition, the global perspective field may include, but is not limited to an age **1260**, popularity **1264**, malware determination **1268**, and/or IP/URL information **1272**, where IP/URL information may include, but is not limited to a categoryID of the IP/URL, a confidence of such a categoryID, a reputation, geolocation information such as which country the IP/URL is associated with, and/or a threat history associated with the IP/URL. Of course, additional relevant information may be included in the event line **1204** depending on the event.

FIG. **13** generally illustrates exemplary user interfaces, or dashboards **1304**, **1332**, and **1336**, which may be displayed at an end recipient **220** to a user at an user output device **412**. The user interface generator **444** may create the one or more dashboards **1304**, **1332**, and **1336** to provide information from an event to a user of the end recipient **220**. Accordingly, the dashboards **1304**, **1332**, and **1336** may display correlated events collected by a sensor agent **204**. Such correlated events may comprise File System/Disk activity **1308**, network activity **1312**, and registry activity **1316** on the endpoint along with the global community data and IP and domain reputation data **1324** to provide a complete point of view of the computing system **204** activity and further provide a full meaning to every single reported event, including at least one of an originating process, an Event Type, a Target, a Victim, and a Actor. Accordingly, spotting and highlighting behaviours in the event data from computing systems **204** may potentially assist in finding and determining whether such activity is malicious or not.

Moreover, there may exist multiple dashboards **1304**, **1332**, and **1336** each of which provide a specific heuristic that targets a specific potential malicious behaviour. For example, a first dashboard **1326** generally looks for executables that have been found and executed from specific critical paths in the computing system **204**; those paths that are known to host malware executables (browser's cache, temporary files folder, system32 folder, auto start folder and so on). The dashboard **1326** may filter those files determined as good, unknown and bad from global community information. For unknown files, the dashboard may show their popularity at the global community, i.e. on how many computing systems **204** they have been seen—and if they establish outbound internet connections. If such files connect to the Internet, the dashboard **1332** may trace down and identify the specific connection and geo-locate the destination address on a world map.

In another embodiment, a separate dashboard **1336** may analyze outbound connections from the analyzed endpoints. Accordingly, this dashboard may filter out and highlight connections to high-risk countries, i.e. those countries known to host malicious servers, and retrieve software that actually established the suspicious connections. The dashboard **1336** may also provide another point of view; that is, the dashboard **1336** may highlight all connections to specific IPs and domains that have been set up recently (e.g. in the last 60 days). All connections may be shown on a map view and the contacted IPs and domains are crosschecked with a database to determine their reputation and website category (if known).

As another example, a dashboard **1340** may highlight potential exploits detected on the endpoint by monitoring the most common software usually installed on an average computing system **204** and targeted by exploit attacks. The dashboard **1340** may generally identify software like Java, Adobe Acrobat, Microsoft Office, Flash Player and monitor if such software is dropping executable files on the system. Accordingly, if the software behaves in this manner, such a behaviour should be considered a suspicious behaviour, as software should not be dropping any executable files on the system. If such an occurrence is detected, the dashboard **1340** may analyze the dropped executable for its popularity in the global community and then identify potential outbound connections established by the just-dropped executable. If an outbound connection is found, the dashboard **1340** retrieves the destination IP and renders, or draws, the location of the IP address on a map view, such as a world map.

As another example, a separate dashboard **1344** may highlight those new processes created on the monitored computing systems **204** that are undetermined and have only been recently seen in the global community. For every process, a lifetime scheme is assembled which allows the administrator to understand when the undetermined file firstly landed on the endpoint and for how long time it has been active on the system. This dashboard **1344** may further provide specific events triggered by the suspicious process while active on the system.

As another example, a dashboard **1348** may highlight system processes that have been found to establish concurrent connections to different IPs in less than a specific timeframe and/or threshold, such as one second. If found, the system processes may be suspect and deemed to host malicious code; such malicious code having been injected into the website. For every established connection, the dashboard may trace and identify the contacted country and the specific IP/domain. As another example, a dashboard **1348** may be generated by the user interface generator **448** such the dashboard monitors activities on remote storage services like Dropbox, showing the filename and size of the files being saved on a remote server. The dashboard **1348** may further monitor and identify sensitive or classified company data that may be in the process of being stolen or placed (i.e. saved, stored, etc.) on a non-approved storage location.

In the foregoing description, for the purposes of illustration, methods were described in a particular order. It should be appreciated that in alternate embodiments, the methods may be performed in a different order than that described. It should also be appreciated that the methods described above may be performed by hardware components or may be embodied in sequences of machine-executable instructions, which may be used to cause a machine, such as a general-purpose or special-purpose processor or logic circuits programmed with the instructions to perform the methods. These machine-executable instructions may be stored on one or more machine readable mediums, such as CD-ROMs or other type of optical disks, floppy diskettes, ROMs, RAMs, EPROMs, EEPROMs, magnetic or optical cards, flash memory, or other types of machine-readable mediums suitable for storing electronic instructions. Alternatively, the methods may be performed by a combination of hardware and software.

Furthermore, embodiments may be implemented by hardware, software, firmware, middleware, microcode, hardware description languages, or any combination thereof. When implemented in software, firmware, middleware or microcode, the program code or code segments to perform the necessary tasks may be stored in a machine readable medium such as storage medium. A processor(s) may perform the necessary tasks. A code segment may represent a procedure, a function, a subprogram, a program, a routine,

a subroutine, a module, a software package, a class, or any combination of instructions, data structures, or program statements. A code segment may be coupled to another code segment or a hardware circuit by passing and/or receiving information, data, arguments, parameters, or memory contents. Information, arguments, parameters, data, etc. may be passed, forwarded, or transmitted via any suitable means including memory sharing, message passing, token passing, network transmission, etc.

The foregoing discussion of the invention has been presented for purposes of illustration and description. Further, the description is not intended to limit the invention to the form disclosed herein. Consequently, variations and modifications commensurate with the above teachings, within the skill or knowledge of the relevant art, are within the scope of the present invention. The embodiments described hereinabove are further intended to explain the best mode presently known of practicing the invention and to enable others skilled in the art to utilize the invention in such or in other embodiments and with various modifications required by the particular application or use of the invention. It is intended that the appended claims be construed to include alternative embodiments to the extent permitted by the prior art.

What is claimed is:

1. A method comprising:

gathering one or more events defining an action of a first object acting on a target;

generating a contextual state for at least one of the one or more events by correlating the at least one event to an originating object, the contextual state including an indication of the originating object of the first object and an indication of at least one of a device on which the first object is executed and a user associated with the first object;

obtaining a global perspective for the at least one event by obtaining information associated with one or more of the first object and the originating object, the information including at least one of age, popularity, a determination as to whether the first object is malware, a determination as to whether the originating object is malware, Internet Protocol (IP) Address, and Uniform Resource Locator (URL) information, wherein the global perspective for one or more related events to the at least one event across a network;

assembling an event line including details associated with the at least one event, the details including information uniquely identifying the first object, the action of the first object, the target, and the originating object; and

transmitting the assembled event line.

2. The method of claim **1**, wherein the information associated with the one or more of the first object and the originating object is obtained from a community database; and

wherein the event line is transmitted to an end recipient.

3. The method of claim **1**, further comprising:

obtaining an event priority based on one or more of the generated context and the global perspective for the at least one event, wherein the event line further includes the event priority for the at least one event.

4. The method of claim **1**, wherein the at least one event includes information uniquely identifying the object performing the action, information uniquely identifying the action being performed, and information uniquely identifying the target upon which the act is being performed.

5. The method of claim **4**, further comprising:

determining whether a global perspective for the at least one event has been cached;

when the global perspective for the at least one event has been cached, determining whether a time-to-live (TTL) value has been exceeded; and

when the global perspective for the at least one event has not been cached, obtaining a global perspective for the at least one event from a community database.

6. The method of claim **5**, further comprising:

determining whether the cached global perspective for the at least one event has expired; and

when the global perspective for the at least one event has expired, obtaining a global perspective for the at least one event from a community database.

7. The method of claim **4**, further comprising:

heuristically displaying information identifying the first object performing the action, information identifying the action being performed, information identifying the target upon which the act is being performed, and the global perspective according to one or more of a location associated with the first object; at least one characteristic of an outbound connection associated with the first object; a dropped executable file; an age of the first object; and a number of concurrent connections associated with the first object.

8. A system for providing forensic visibility into a system, comprising:

a device including:

a communication interface;

a processor;

data storage; and

a sensor agent stored on the data storage that is executable by the processor, wherein the sensor agent is operable to:

gather one or more events defining an action of a first object acting on a target;

generate a contextual state for at least one of the one or more events by correlating the at least one event to an originating object, the contextual state including an indication of the originating object of the first object and an indication of at least one of a device on which the first object is executed and a user associated with the first object;

obtain a global perspective for the at least one event by obtaining information associated with one or more of the first object and the originating object, the information including at least one of age, popularity, a determination as to whether the first object is malware, a determination as to whether the originating object is malware, Internet Protocol (IP) Address, and Uniform Resource Locator (URL) information, wherein the global perspective for one or more related events to the at least one event across a network;

assemble an event line including details associated with the at least one event, the details including information uniquely identifying each of the first object, the action of the first object, the target, and the originating object; and

transmit the assembled event line utilizing the communication interface.

9. The system of claim **8**, wherein the at least one of the one or more events is executed by the processor.

10. The system of claim **8**, further comprising:

a global perspective information server; and

an end recipient device, wherein the information associated with the one or more of the first object and the

originating object is obtained from the global perspective information server and the sensor agent transmits the event line to the end recipient device.

11. The system of claim **8**, wherein the sensor agent is further operable to: cache the one or more events; and asynchronously consolidate and normalize the one or more events.

12. The system of claim **8**, wherein the at least one event includes information uniquely identifying the object performing the action, information uniquely identifying the action being performed, and information uniquely identifying the target upon which the act is being performed.

13. The system of claim **12**, wherein the sensor agent is further operable to:

determine whether the global perspective for the at least one event has been cached;

when the global perspective for the at least one event has been cached, determining whether a time-to-live (TTL) value has been exceeded; and

when the global perspective for the at least one event has not been cached, obtain a global perspective for the at least one event from a community database.

14. The system of claim **13**, wherein the sensor agent is further operable to:

determine whether the cached global perspective for the at least one event has expired; and

when the global perspective for the at least one event has expired, obtain a global perspective for the at least one event from a community database.

15. The system of claim **12**, wherein the end recipient heuristically displays information identifying the first object performing the action, information identifying the action being performed, information identifying the target upon which the act is being performed, and the global perspective according to one or more of a location associated with the first object; at least one characteristic of an outbound connection associated with the first object; a dropped executable file; an age of the first object; and a number of concurrent connections associated with the first object.

16. A system for providing forensic visibility into a system, comprising:

a device including:

a communication interface;

a processor;

data storage; and

a sensor agent stored on the data storage that is executable by the processor, wherein the sensor agent is operable to:

gather one or more events defining an action of a first object acting on a target;

generate a contextual state for at least one of the one or more events by correlating the at least one event to an originating object, the contextual state including an indication of the originating object of the first object and an indication of at least one of a device on which the first object is executed and a user associated with the first object;

assemble an event line including details associated with the at least one event, the details including information uniquely identifying each of the first object, the action of the first object, the target, the originating object, and global perspective information comprising information about one or more related events to the one or more events across a network; and

transmit the assembled event line utilizing the communication interface.

17. The system of claim **16**, further comprising a global perspective information server including:

a processor;

a communication interface;

data storage; and

a global perspective module, wherein the global perspective module is operable to:

receive the event line from the device;

obtain a global perspective for the at least one event by obtaining information associated with one or more of the first object and the originating object, the information including at least one of age, popularity, a determination as to whether the first object is malware, a determination as to whether the originating object is malware, Internet Protocol (IP) Address, and Uniform Resource Locator (URL) information;

assemble a second event line including details associated with the at least one event, the details including information uniquely identifying each of the first object, the action of the first object, the target, and the originating object; and

transmit the assembled second event line utilizing the communication interface to an end recipient.

18. The system of claim **17**, wherein the end recipient heuristically displays information identifying the first object performing the action, information identifying the action being performed, information identifying the target upon which the act is being performed, and the global perspective according to one or more of a location associated with the first object; at least one characteristic of an outbound connection associated with the first object; a dropped executable file; an age of the first object; and a number of concurrent connections associated with the first object.

19. The system of claim **16**, further comprising a bridge including:

a processor;

a communication interface; and

a cached event store, wherein the cached event store is operable to:

receive a first request for a global perspective for the at least one event;

receive a second request for a global perspective for another event, the second request originating from another device;

obtain a first global perspective for the at least one event by obtaining information associated with one or more of the first object and the originating object, the information including at least one of age, popularity, a determination as to whether the first object is malware, a determination as to whether the originating object is malware, Internet Protocol (IP) Address, and Uniform Resource Locator (URL) information;

obtain a second global perspective for the another event;

provide the first global perspective to the device; and

provide the second global perspective to the another device.

* * * * *

Exhibit 4

U 8164950

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

October 07, 2021

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THIS OFFICE OF:

U.S. PATENT: *10,257,224*
ISSUE DATE: *April 09, 2019*

By Authority of the
Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office

R GLOVER
Certifying Officer

US010257224B2

(12) **United States Patent**
Jaroch et al.

(10) **Patent No.: US 10,257,224 B2**
(45) **Date of Patent: *Apr. 9, 2019**

(54) **METHOD AND APPARATUS FOR PROVIDING FORENSIC VISIBILITY INTO SYSTEMS AND NETWORKS**

(71) Applicant: **WEBROOT INC.**, Broomfield, CO (US)

(72) Inventors: **Joseph Jaroch**, Deer Park, IL (US); **Jacques Etienne Erasmus**, Belper (GB); **Paul Barnes**, Derby (GB); **Johannes Mayr**, Linz (AT); **Michael Leidesdorff**, Niwot, CO (US); **Marco Giuliani**, Bastia Umbra (IT); **Christopher Jon Williams**, Derbyshire (GB); **Chad Edward Bacher**, Arvada, CO (US)

(73) Assignee: **Webroot Inc.**, Broomfield, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **15/437,275**

(22) Filed: **Feb. 20, 2017**

(65) **Prior Publication Data**

US 2017/0163660 A1 Jun. 8, 2017

Related U.S. Application Data

(63) Continuation of application No. 14/270,069, filed on May 5, 2014, now Pat. No. 9,578,045.

(Continued)

(51) **Int. Cl.**

G06F 21/00	(2013.01)
H04L 29/06	(2006.01)
G06F 21/56	(2013.01)

(52) **U.S. Cl.**

CPC ***H04L 63/145*** (2013.01); ***G06F 21/56*** (2013.01); ***H04L 63/1416*** (2013.01); ***H04L 63/1425*** (2013.01)

(58) **Field of Classification Search**

CPC .. H04L 63/1416; H04L 63/20; H04L 63/1425; H04L 63/1441; H04L 63/1408;

(Continued)

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,895,649 B1	2/2011	Brook et al.
9,578,045 B2	2/2017	Jaroch et al.

(Continued)

FOREIGN PATENT DOCUMENTS

WO	2006094228	9/2006
WO	2012110501	8/2012

OTHER PUBLICATIONS

European Extended Search Report in Application 14791882.5, dated Nov. 11, 2016, 8 pages

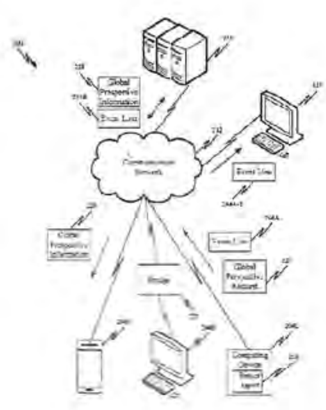
(Continued)

Primary Examiner — Ghazal B Shehni

(57) **ABSTRACT**

Methods and systems for providing forensic visibility into systems and networks are provided. More particularly, a sensor agent may receive events defining an action of a first object acting on a target. The object, the event, and the target are then correlated to at least one originating object such that an audit trail for each individual event is created. A global perspective indicating an age, popularity, a determination as to whether the object may be malware, and IP/URL information associated with the event may then be applied to at least one of the object, the event, the target, and the originating object. A priority may then be determined and assigned to the event based on at least the global perspective. An event line containing event information is then trans-

(Continued)



Copy provided by USPTO from the PIRS Image Database on 10-05-2021

mitted to an end recipient where the information may be heuristically displayed.

20 Claims, 15 Drawing Sheets

Related U.S. Application Data

(60) Provisional application No. 61/819,470, filed on May 3, 2013.

(58) **Field of Classification Search**
CPC ... H04L 63/1433; H04L 63/145; H04L 63/14; H04L 9/3271; H04L 12/2602
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2006/0095963	A1	5/2006	Crosby et al.	
2008/0016570	A1*	1/2008	Capalik	H04L 63/1408 726/23
2008/0244694	A1	10/2008	Neystadt et al.	
2009/0300769	A1	12/2009	Srinivasa et al.	
2012/0072725	A1	3/2012	Fanton et al.	
2012/0260340	A1	10/2012	Morris et al.	
2013/0247201	A1*	9/2013	Alperovitch	H04L 63/14 726/24
2013/0298244	A1*	11/2013	Kumar	G06F 21/52 726/25

OTHER PUBLICATIONS

PCT International Preliminary Report on Patentability in PCT/US2014/036832, dated Nov. 3, 2015, 8 pages.
PCT International Search Report in PCT/US2014/036832, dated Sep. 29, 2014, 9 pages.
U.S. Appl. No. 14/270,069, Amendment and Response filed Mar. 16, 2016, 16 pages.
U.S. Appl. No. 14/270,069, Amendment and Response filed Aug. 31, 2016, 13 pages.
U.S. Appl. No. 14/270,069, Notice of Allowance dated Oct. 5, 2016, 8 pages.
U.S. Appl. No. 14/270,069, Office Action dated Sep. 16, 2015, 12 pages.
U.S. Appl. No. 14/270,069, Office Action dated May 31, 2016, 11 pages.
European Office Action in Application 14791882.5, dated Dec. 8, 2017, 6 pages.
European Communication in Application 14791882.5, dated Nov. 16, 2018, 5 pages.

* cited by examiner

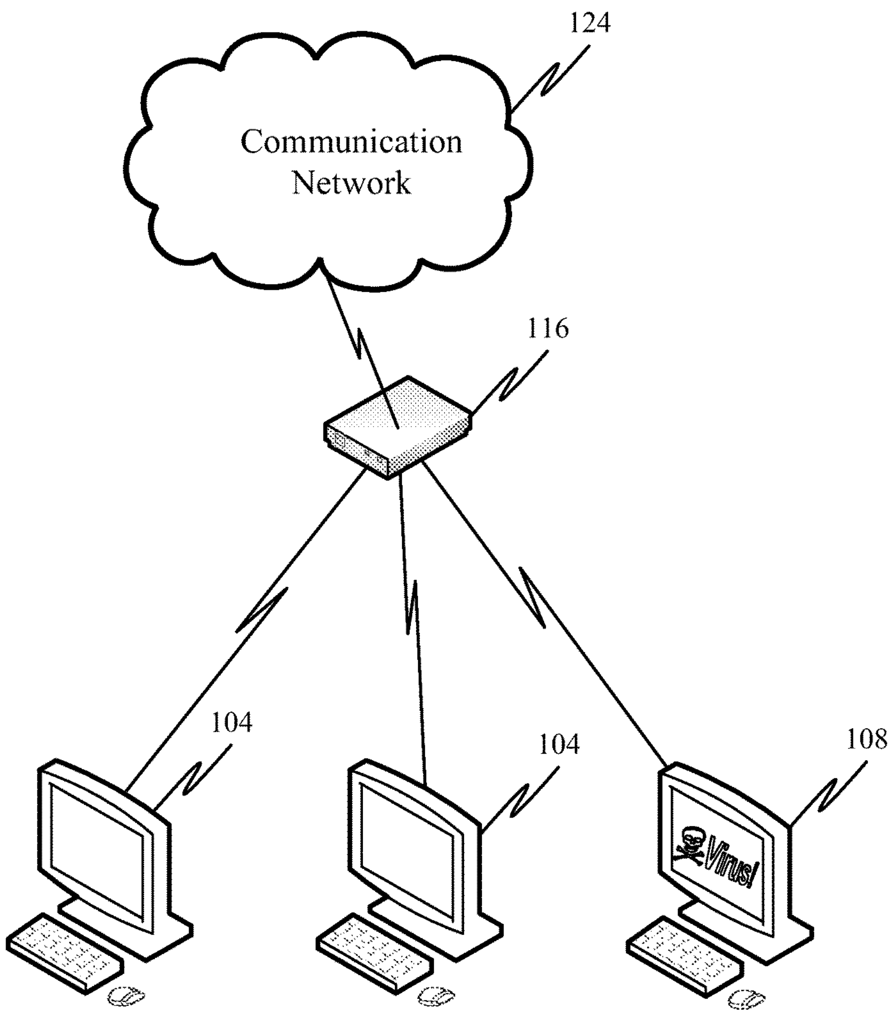


Figure 1

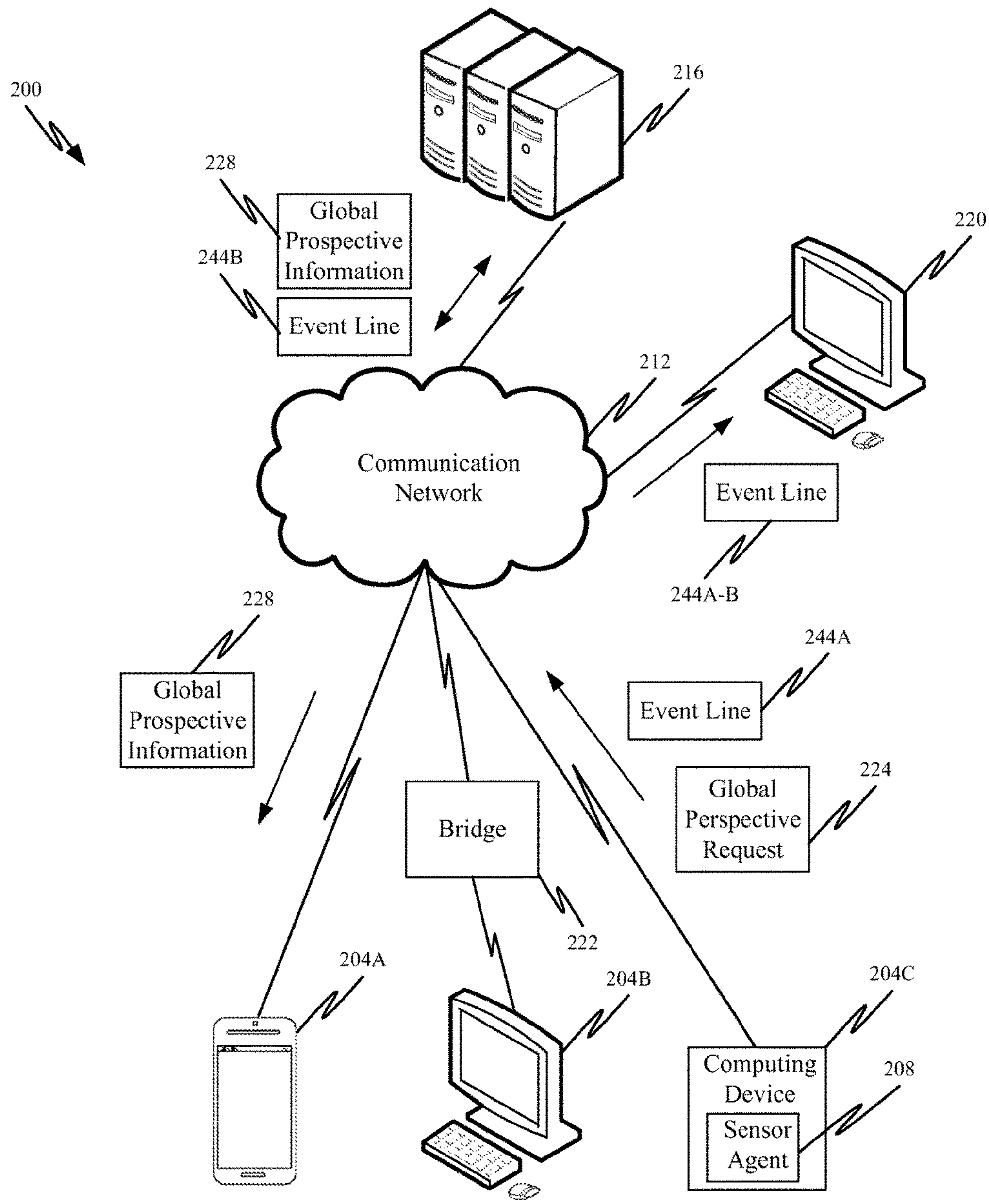

200
216
228
Global Prospective Information
244B
Event Line
220
212
Communication Network
Event Line
244A-B
228
Global Prospective Information
244A
Event Line
Bridge
Global Perspective Request
224
222
204A
204B
204C
Computing Device
Sensor Agent
208

Figure 2

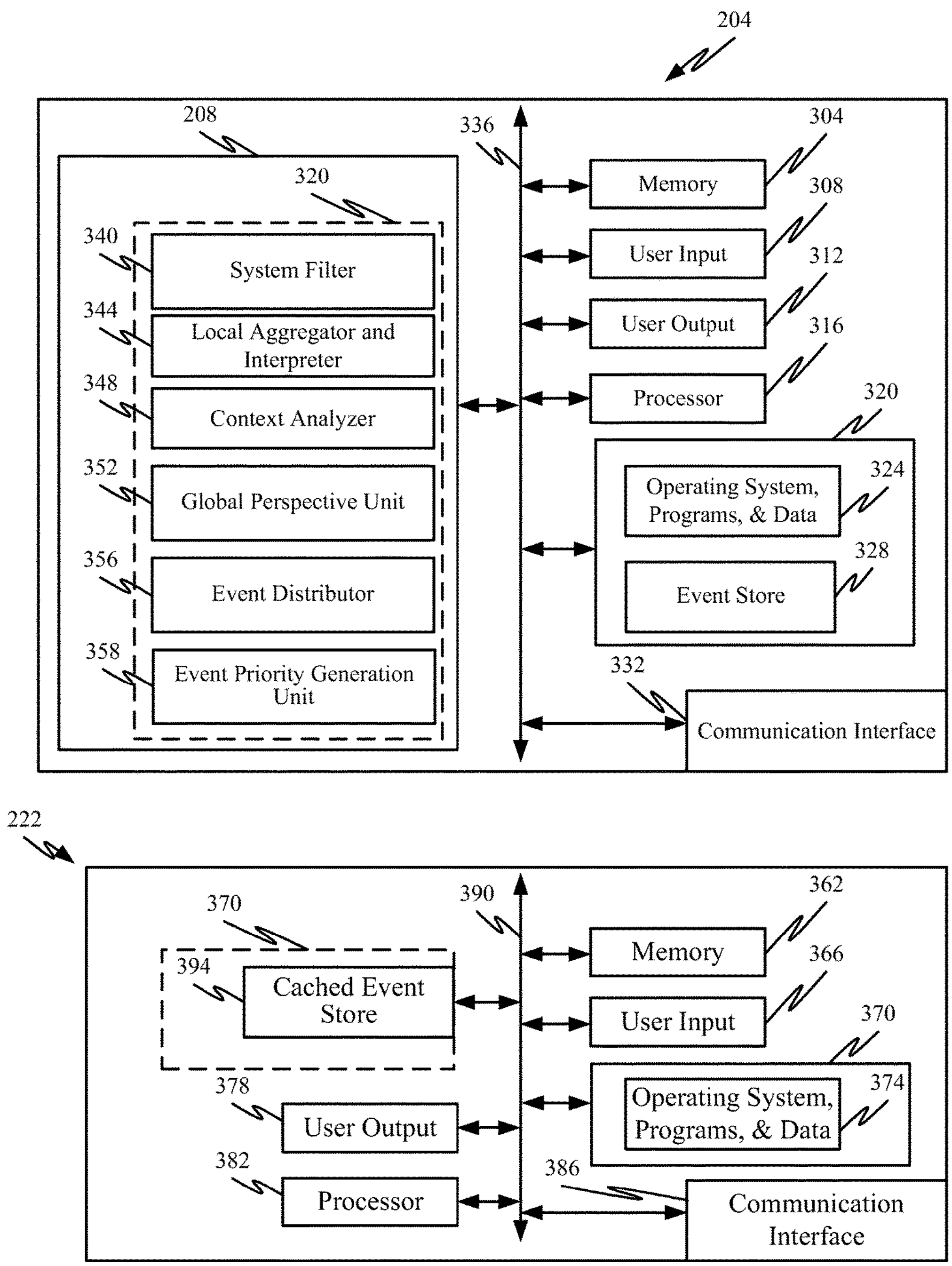

204
208
336
304
320
Memory
308
340
System Filter
User Input
312
344
Local Aggregator and Interpreter
User Output
316
348
Context Analyzer
Processor
320
352
Global Perspective Unit
Operating System, Programs, & Data
324
328
356
Event Distributor
Event Store
358
332
Event Priority Generation Unit
Communication Interface
222
390
362
370
Memory
394
Cached Event Store
366
User Input
370
378
374
User Output
Operating System, Programs, & Data
382
386
Processor
Communication Interface

Figure 3

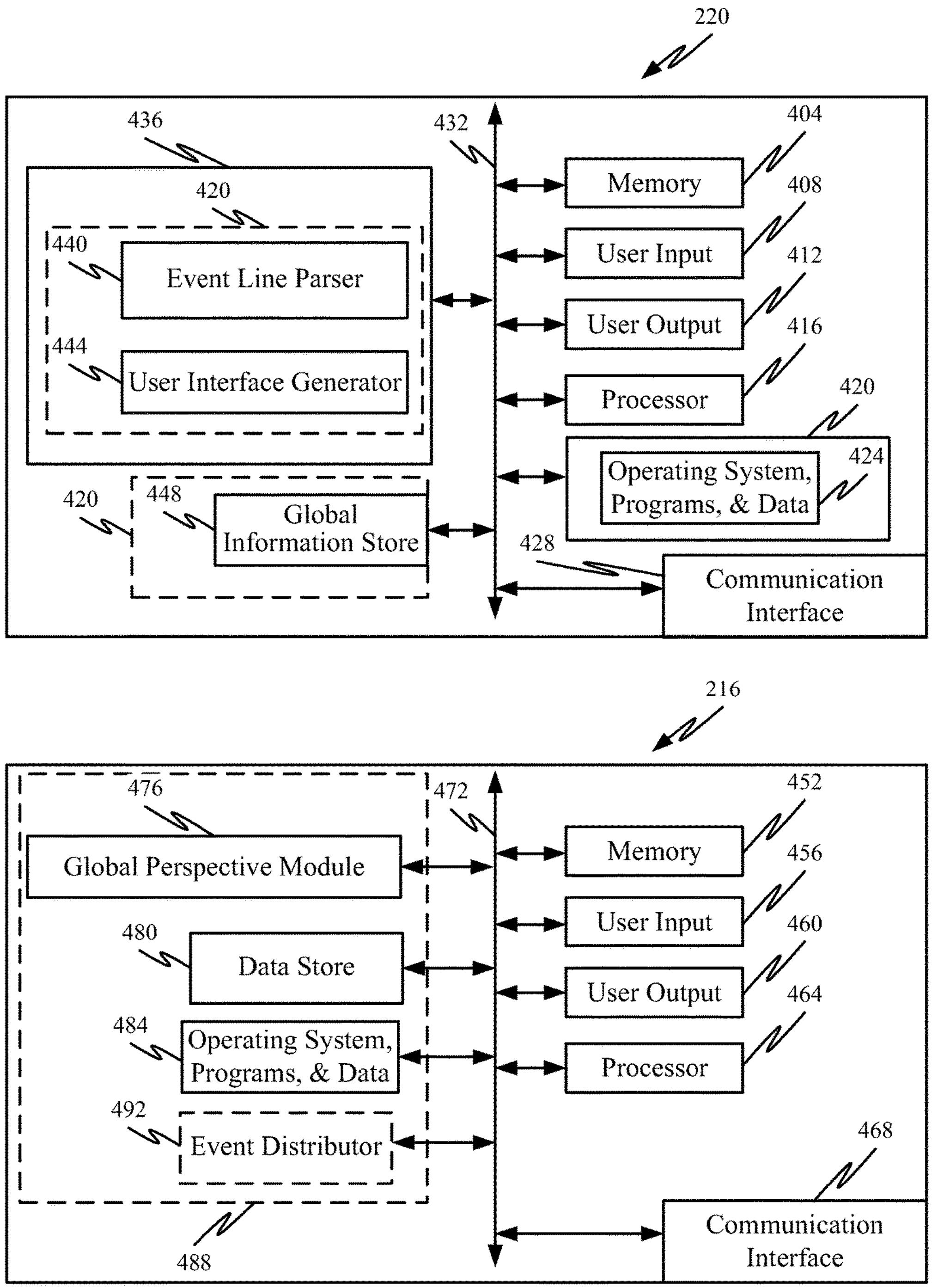

220
436
432
404
420
Memory
408
440
User Input
Event Line Parser
412
User Output
444
416
User Interface Generator
Processor
420
424
Operating System, Programs, & Data
420
448
Global Information Store
428
Communication Interface
216
476
472
452
Memory
Global Perspective Module
456
User Input
480
460
Data Store
User Output
484
464
Operating System, Programs, & Data
Processor
492
468
Event Distributor
Communication Interface
488

Figure 4

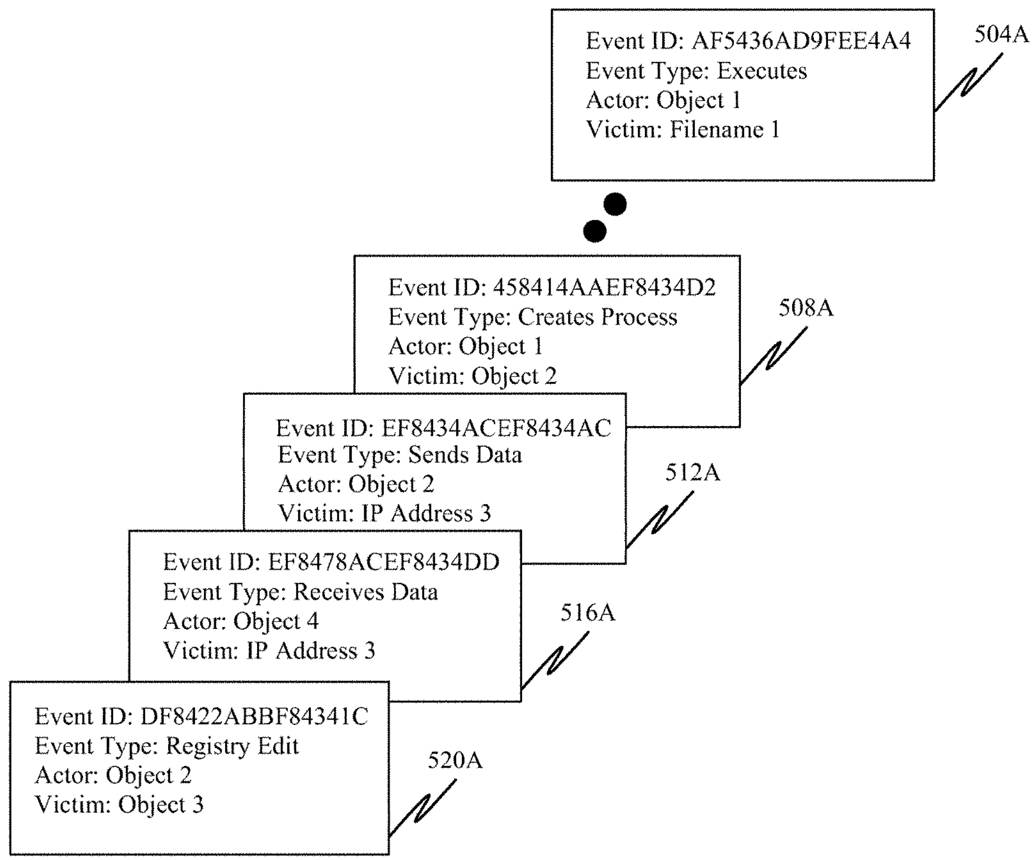

Event ID: AF5436AD9FEE4A4
Event Type: Executes
Actor: Object 1
Victim: Filename 1
504A
Event ID: 458414AAEF8434D2
Event Type: Creates Process
Actor: Object 1
Victim: Object 2
508A
Event ID: EF8434ACEF8434AC
Event Type: Sends Data
Actor: Object 2
Victim: IP Address 3
512A
Event ID: EF8478ACEF8434DD
Event Type: Receives Data
Actor: Object 4
Victim: IP Address 3
516A
Event ID: DF8422ABBF84341C
Event Type: Registry Edit
Actor: Object 2
Victim: Object 3
520A

Figure 5A

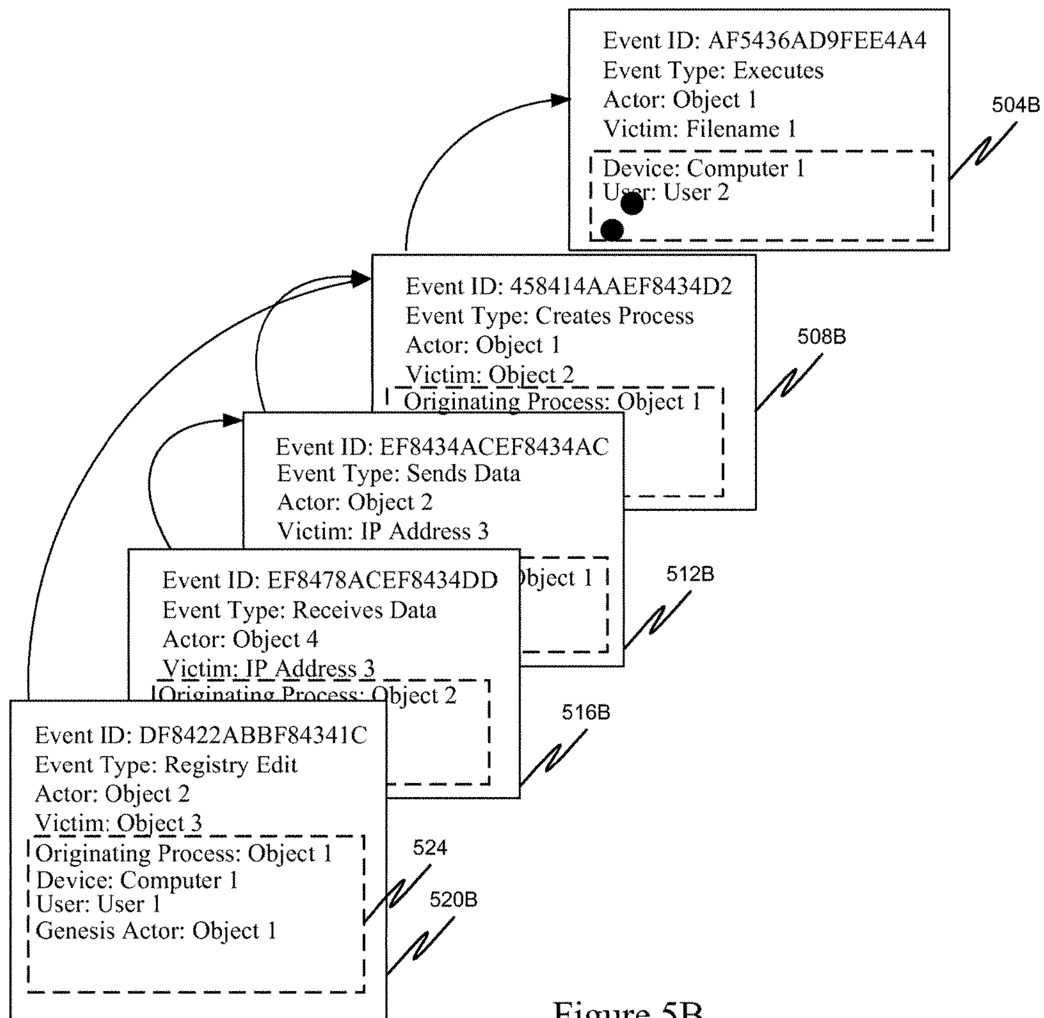

Event ID: AF5436AD9FEE4A4
Event Type: Executes
Actor: Object 1
Victim: Filename 1
Device: Computer 1
User: User 2
504B
Event ID: 458414AAEF8434D2
Event Type: Creates Process
Actor: Object 1
Victim: Object 2
Originating Process: Object 1
508B
Event ID: EF8434ACEF8434AC
Event Type: Sends Data
Actor: Object 2
Victim: IP Address 3
512B
Event ID: EF8478ACEF8434DD
Event Type: Receives Data
Actor: Object 4
Victim: IP Address 3
Originating Process: Object 2
516B
Event ID: DF8422ABBF84341C
Event Type: Registry Edit
Actor: Object 2
Victim: Object 3
Originating Process: Object 1
Device: Computer 1
User: User 1
Genesis Actor: Object 1
524
520B

Figure 5B

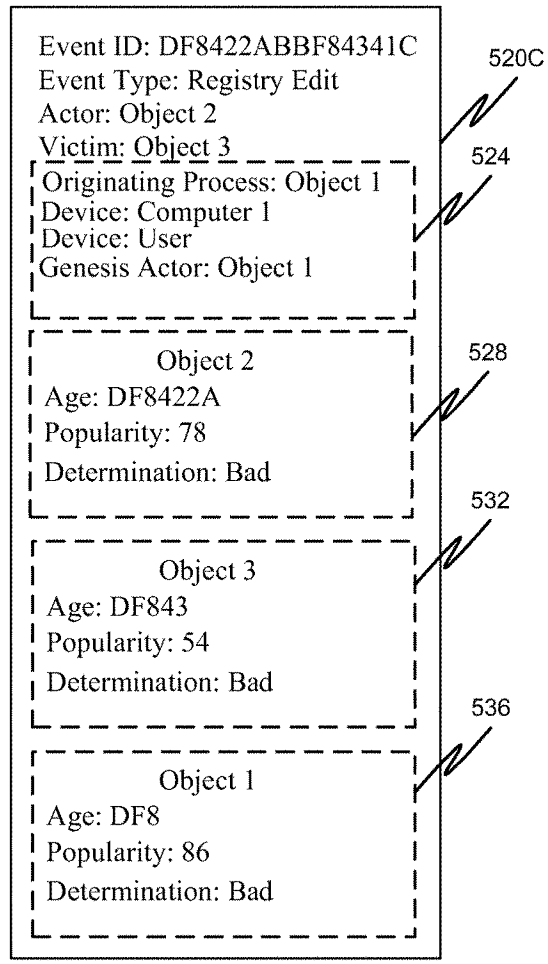
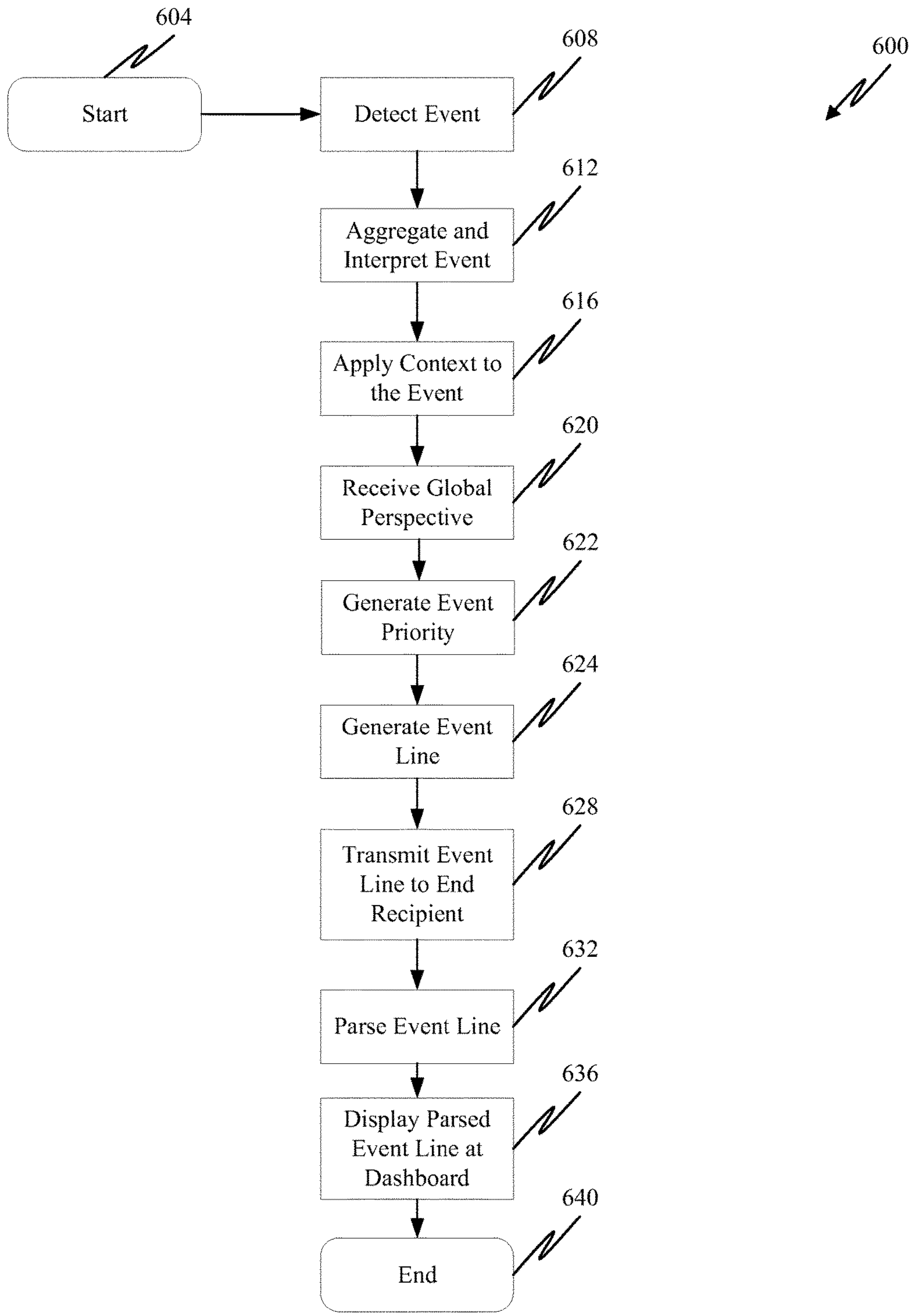

Event ID: DF8422ABBF84341C
Event Type: Registry Edit
Actor: Object 2
Victim: Object 3
Originating Process: Object 1
Device: Computer 1
Device: User
Genesis Actor: Object 1
Object 2
Age: DF8422A
Popularity: 78
Determination: Bad
Object 3
Age: DF843
Popularity: 54
Determination: Bad
Object 1
Age: DF8
Popularity: 86
Determination: Bad
520C
524
528
532
536

Figure 5C



604
608
600
Start
Detect Event
612
Aggregate and Interpret Event
616
Apply Context to the Event
620
Receive Global Perspective
622
Generate Event Priority
624
Generate Event Line
628
Transmit Event Line to End Recipient
632
Parse Event Line
636
Display Parsed Event Line at Dashboard
640
End

Figure 6

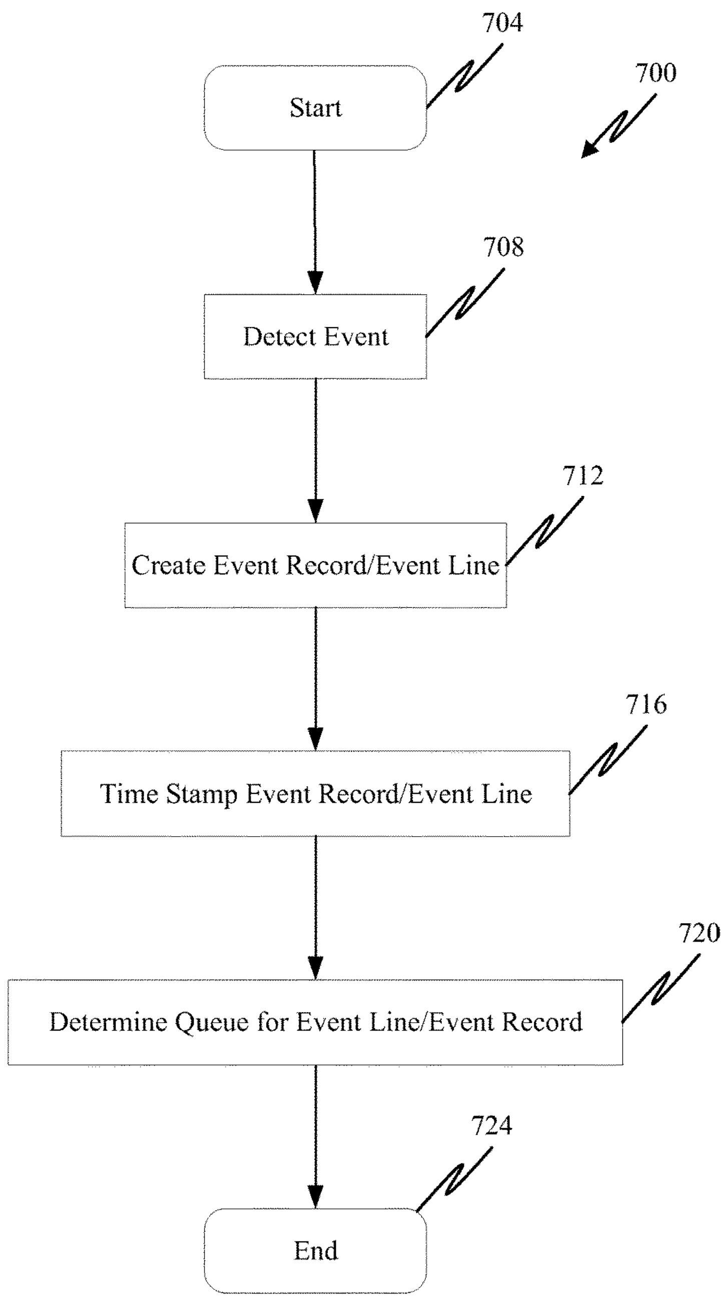
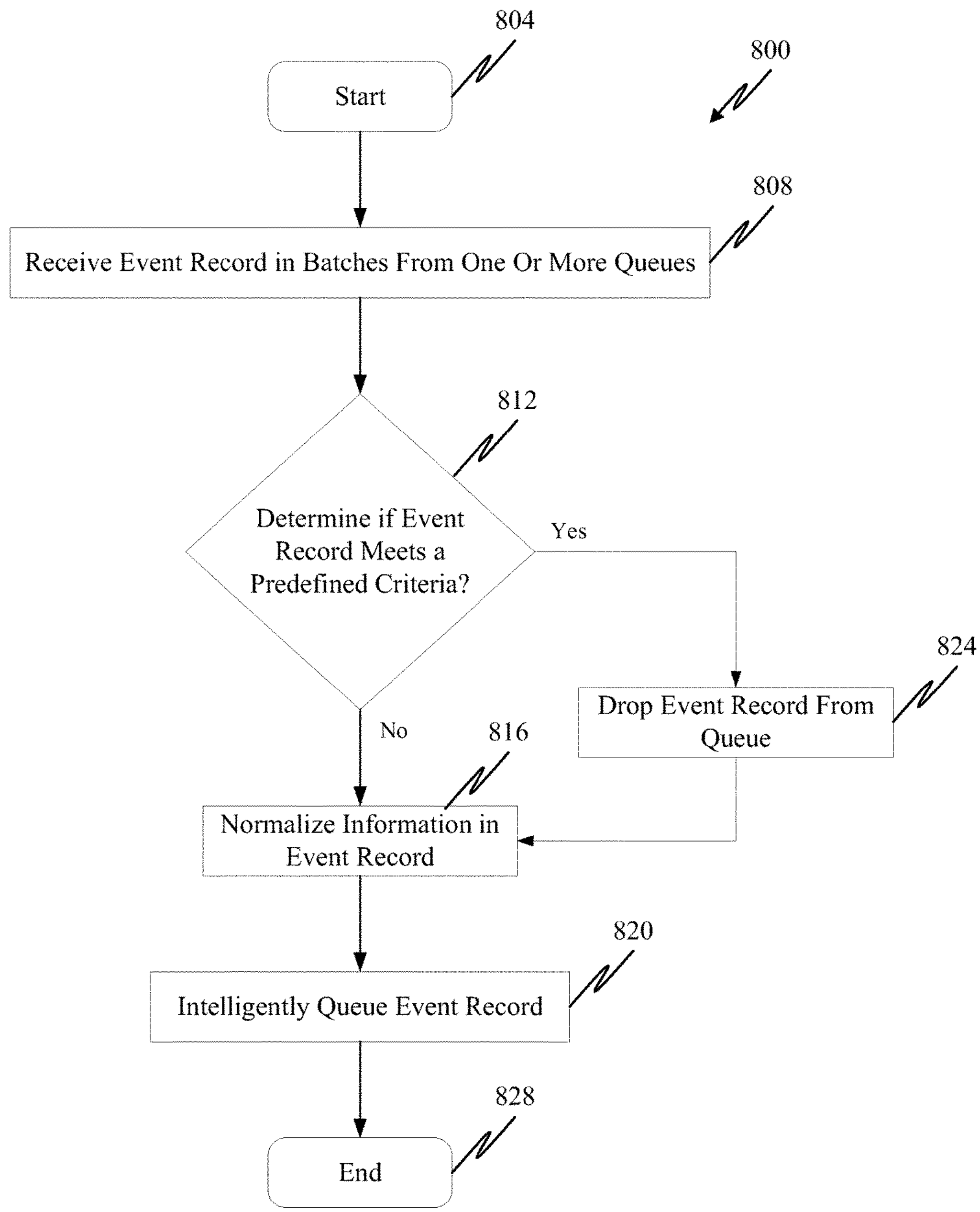

704
700
Start
708
Detect Event
712
Create Event Record/Event Line
716
Time Stamp Event Record/Event Line
720
Determine Queue for Event Line/Event Record
724
End

Figure 7



804
800
Start
808
Receive Event Record in Batches From One Or More Queues
812
Determine if Event Record Meets a Predefined Criteria?
Yes
824
Drop Event Record From Queue
No
816
Normalize Information in Event Record
820
Intelligently Queue Event Record
828
End

Figure 8

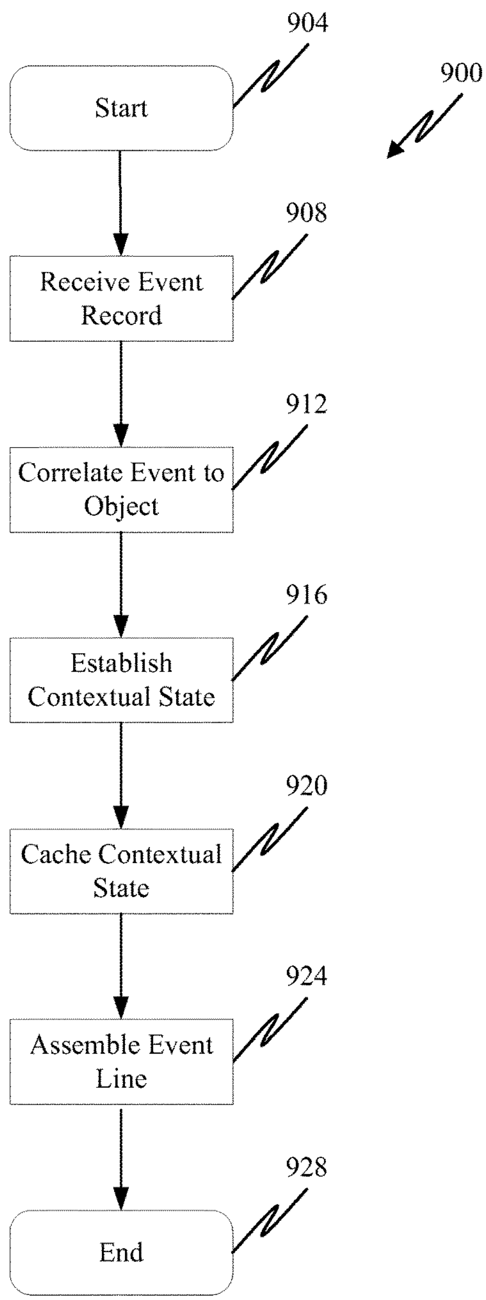

900
904
Start
908
Receive Event Record
912
Correlate Event to Object
916
Establish Contextual State
920
Cache Contextual State
924
Assemble Event Line
928
End

Figure 9

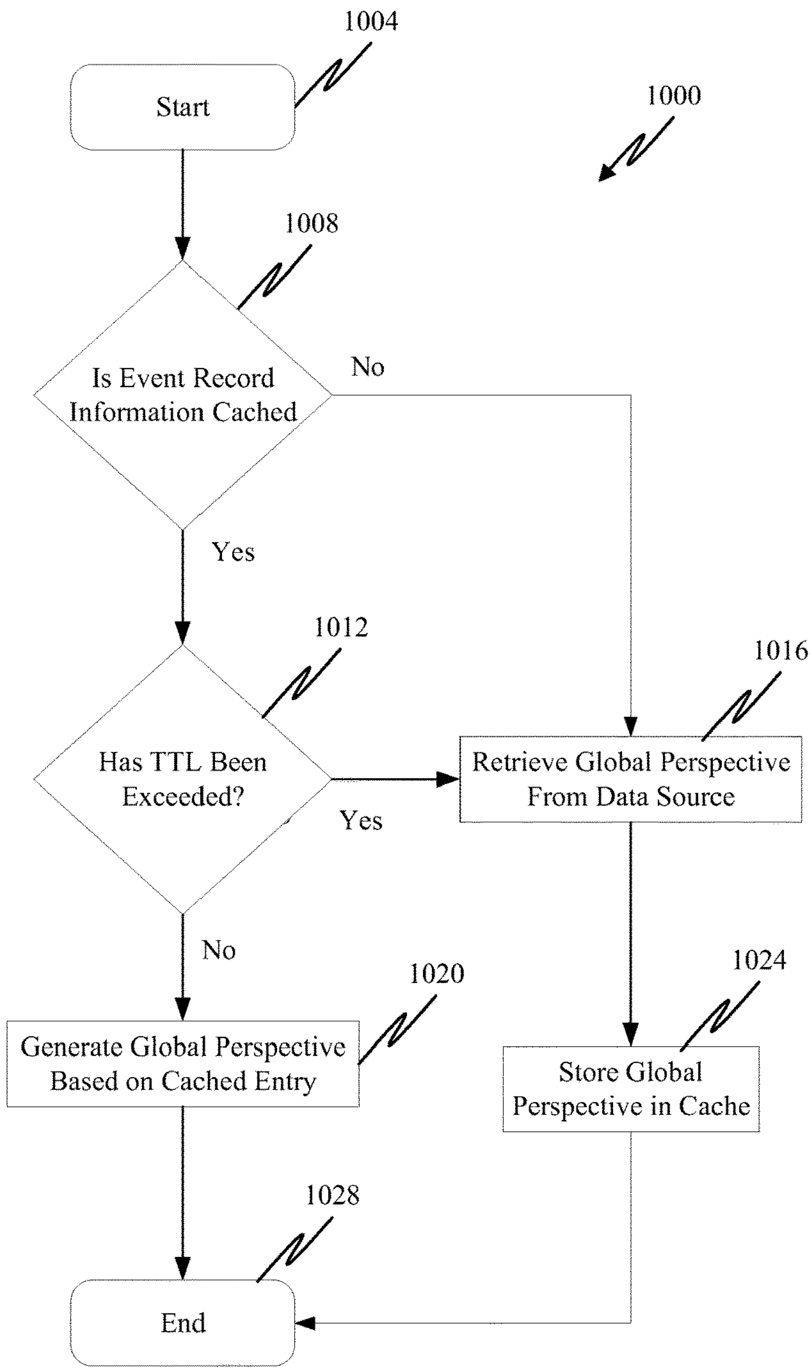

1004
Start
1000
1008
Is Event Record Information Cached
No
Yes
1012
1016
Has TTL Been Exceeded?
Retrieve Global Perspective From Data Source
Yes
No
1020
1024
Generate Global Perspective Based on Cached Entry
Store Global Perspective in Cache
1028
End

Figure 10

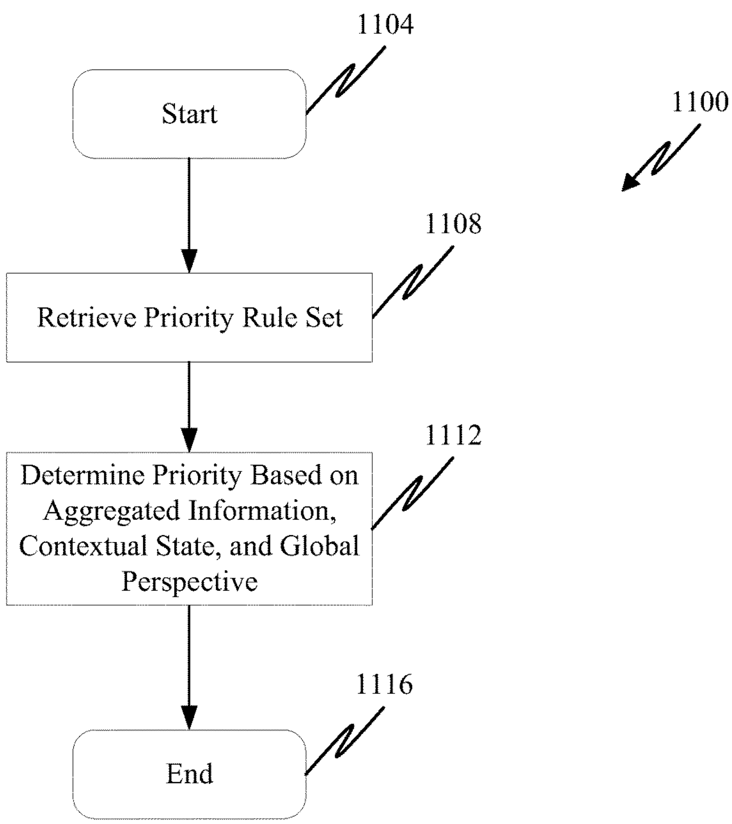

1104
Start
1100
1108
Retrieve Priority Rule Set
1112
Determine Priority Based on Aggregated Information, Contextual State, and Global Perspective
1116
End

Figure 11

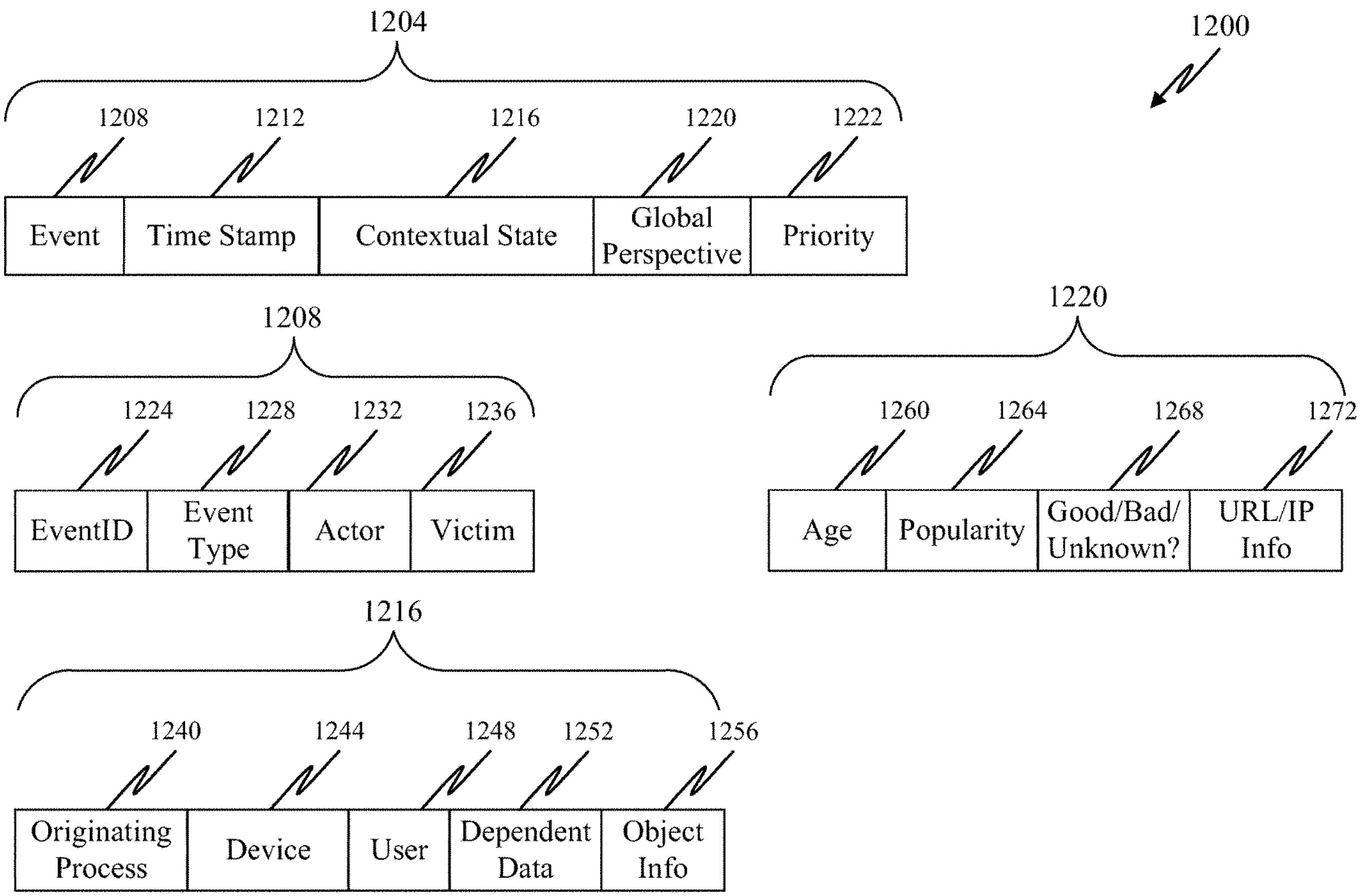

1200
1204
1208
1212
1216
1220
1222
Event
Time Stamp
Contextual State
Global Perspective
Priority
1208
1224
1228
1232
1236
EventID
Event Type
Actor
Victim
1220
1260
1264
1268
1272
Age
Popularity
Good/Bad/ Unknown?
URL/IP Info
1216
1240
1244
1248
1252
1256
Originating Process
Device
User
Dependent Data
Object Info

Figure 12

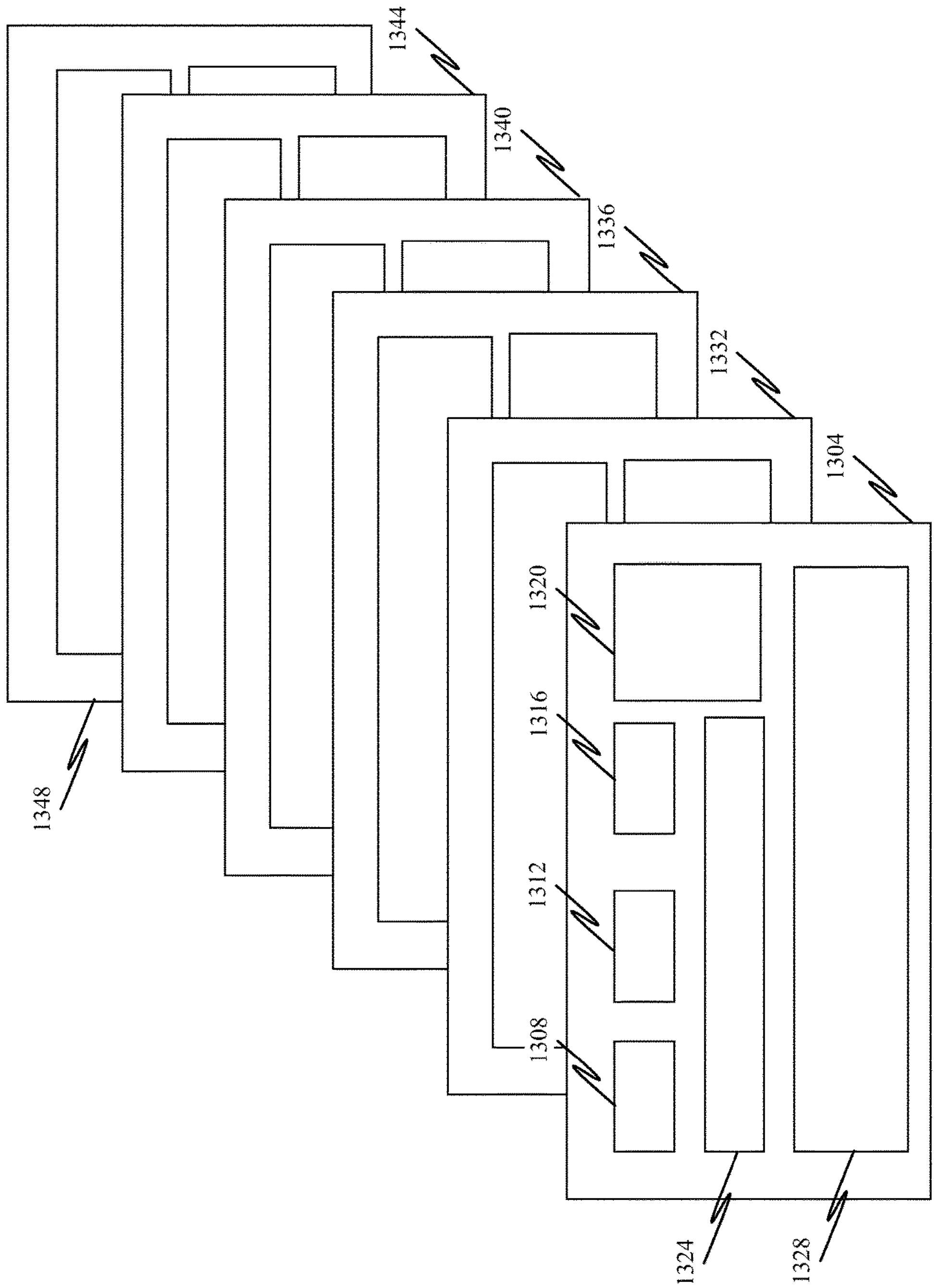


Figure 13

METHOD AND APPARATUS FOR PROVIDING FORENSIC VISIBILITY INTO SYSTEMS AND NETWORKS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation application of U.S. patent application Ser. No. 14/270,069, filed May 5, 2014, which application claims the benefit of U.S. Provisional Patent Application Ser. No. 61/819,470, filed May 3, 2013, the entire disclosures of which are hereby incorporated herein by reference in their entireties.

FIELD

Methods and apparatuses for providing forensic visibility are described. More particularly, methods and systems for providing forensic visibility into client systems having full audit trails are described.

BACKGROUND

Network forensics generally relates to the capture, recording, and analysis of network events in order to discover the source of security attacks or other problem incidents. Network forensic systems generally utilize one of two approaches: the "catch it as you can approach" and the"stop, look, and listen approach." "Catch it as you can" systems immediately write the packets to a disk file, buffering in memory as necessary, and perform analysis in batches. "Stop, look and listen" systems analyze the packets in memory, perform rudimentary data analysis and reduction, and write selected results to disk or to a database over the network.

However, existing network forensic tools, such as network sniffers and packet capturing tools, passively collect information about network traffic. Such tools may be able to detect and capture information associated with communication sessions, such as hostnames, ports, protocols, IP addresses, etc.; however, such tools tend to receive network events outside of a host or system. That is, a network forensic tool, such as a network sniffer, may reside at a network edge and may receive network events concerning the network from one or more hosts or systems. Accordingly, network edge devices have no ability to establish any sort of context from within the host or system that is generating the event. Therefore, devices residing at a network edge may monitor, record, and or analyze network traffic between the host and the device, but information that may be useful within the host is not obtained.

For example, FIG. **1** generally illustrates a comparative example including computing devices **104** and **108** in communication with a network edge device **116**. The network edge device **116** may be commonly referred to as a "sniffer" which may monitor, capture, and analyze network traffic between the network edge device **116** and the computing devices **104** and **108**. The network edge device **116** may further be in communication with a communication network **124**, such as the internet. Accordingly, the network edge device **116** may monitor, capture, and analyze network traffic between the computing devices **104** and **108** and between the computing devices **104**/**108** and the communication network **124**.

However, as previously mentioned, the network edge device **116** has no ability to monitor, capture, and/or analyze what is occurring within each of the computing device **104** and **108**. In at least one comparative example, computing device **108** may contain or otherwise be infected with one or more pieces of malware. The term "malware" is used herein to refer generally to any executable computer file or, more generally "object", that is or contains malicious code, and thus includes viruses, Trojans, worms, spyware, adware, etc. and the like. Indeed, the network edge device **116** may be able to monitor, capture, and analyze communication traffic including the information, both encrypted and unencrypted, source and destination ports, source and destination IP address, and protocols utilized by the computing device **108**, but the network edge device cannot provide details regarding files modified, registry entries changed, and/or new processes that are created. Accordingly, the network edge device **116** has no way of knowing to what extent the virus may have impacted the computing device **108**, what additional files may have become infected, or what processes and/or objects operating are linked to the virus. What is needed therefore, is a solution that eliminates the guesswork and allows for end-to-end visibility of every event within a system and/or within a network.

SUMMARY

It is, therefore, one aspect of the present disclosure to provide a system and method whereby events occurring within a computing device are captured and additional context and a global perspective is provided for each capture event. For example, a sensor agent may provide visibility into occurrences across an environment, such as a networked environment, to ensure that an administrator is aware of any system changes and data communication in and out of computing devices residing on the network. In this implementation, six components, or modules, may be utilized in the event gathering and processing: low level system filters, a local aggregator and interpreter, a context analyzer, a global perspective module, an event priority module, and an event distributor may be utilized to provide forensic visibility into one or more computing devices of a communication network.

The system filters may be built upon the same or similar technology related to behavior monitoring and collection, as discussed in U.S. application Ser. No. 13/372,375 filed Feb. 13, 2012, "Methods and Apparatus for Dealing with Malware" (US-2012-0260340-A1), which is incorporated herein by reference in its entirety for all that it teaches and for all purposes. These filters intercept system events in a manner such that the operation of the system filter does not impact system performance, a key aspect of this solution. Events at this level tend to be very raw and generally not individually useful—on an average system, tens of millions of events take place every minute and the noise ratio can prevent forensic solutions from being able to provide sufficient value to the end consumer of their data due to the inability to quickly find important events. A product which impacts system performance will have considerably diminished value to an administrator and can negatively affect the results of an analysis undertaken. For example, a system filter that requires a majority of the available computing and system resources may render one or more communication devices unable to perform its primary function and may steer a real-time forensics analysis to an irrelevant analysis of events; irrelevant because such event analysis of information may be based on old and outdated event information. In some embodiments, events are gathered and timestamped accurately and sequentially, establishing a chronological order of system events for consumption at a higher level.

In at least one embodiment, the second component, the local aggregator and interpreter, receives events from the low level system filters, collecting them in batches from the queues in which they are inserted in real-time, and transmits these events to a local service process where the events are held for further analysis. At this stage, intelligent caching may be invoked to drop events which are duplicates of recent events, and some data normalization may be performed to ensure consistency. Caching may be implemented sparingly but in critical areas to ensure redundant events are not continually reported—many applications perform thousands of the same events (network access to the same destination IP address, for example) and transmitting these events into an external log source would create a significant amount of unnecessary noise and potential performance degradation. The cache may integrate a "time to live" concept to prevent applications from hiding their traffic within their noise. The caches may automatically roll over to ensure sufficient space is always available without requiring exorbitant amounts of memory.

In addition, data normalization takes items such as kernel filenames, usermode filenames, process objects, registry objects, and other system structures and normalizes them into a standard format which is easy more easily consumed by automated parsers. Consolidating and normalizing data at this stage allows the data to be handled much more quickly at the higher levels to avoid reprocessing data and potentially missing events just because they were referred to slightly differently (\Device\HarddiskVolume1\Windows\file.exe instead of C:\Windows\file.exe for example). In some embodiments, an aspect of this subsystem provides the ability to process events asynchronously. The subsequent modules which will execute over the data can potentially take a considerable amount of time to complete but delaying the response for each event at this phase would result in a significant system slowdown. Instead, events are intelligently queued in multiple layers to minimize the number of individual threads but still ensure each event is given sufficient time to be independently processed. Because of this structure, there is no performance difference if an event takes one millisecond to be processed or one hour for the full series of routines to complete.

In at least some embodiments, the context analyzer is utilized to ensure that a user, such as a network administrator, is provided the deepest view of the most important data but still has access to the full range of data if they wish to see it. The analysis starts by correlating events to objects (their originating processes, devices, and users), and creating an audit trail for each individual event. This allows an administrator to forensically track back any event which occurs to what triggered it. As previously discussed, conventional products in this space receive events outside of the system (for example, as a network edge device) and therefore have no ability to establish context within the system. The present solution eliminates the guesswork and allows for end-to-end visibility of every event from the bottom of the stack to the top. As contextual state is established, it is cached and aggregated to be quickly queried in subsequent lookups further reducing latency and improving the responsiveness of the system.

After the contextual analysis completes, the sensor agent attempts to establish a global perspective for each event by calling the cloud database using an infrastructure similar to the infrastructure discussed in U.S. application Ser. No. 13/372,375 filed Feb. 13, 2012, "Methods and Apparatus for Dealing with Malware" (US-2012-0260340-A1), but with an additional layer capable of responding in a highly scalable manner which avoids the overhead generally associated with the complex rule processing within the overall database. This response includes information related to the age, popularity, and determination of an object—whether it is known good, bad, or unknown. This data may then be normalized and appended to each relevant portion of the event line. Further lookups are performed for network activity, assessing the reputation and category of each website and IP address as they are contacted.

The event priority module may contain priority logic, or one or more rule sets, to associate a priority to one or more events. The event priority module may utilize information from the local aggregator and interpreter, the context analysis module, and the global perspective module, to determine, or otherwise assign, a priority to an event; such a priority may be based on a rule set. The rule set may be provided by an administration system, backend system, end recipient **220**, or other component of a forensic system.

The event distributor may then batch and transmit the events to the end recipient after the global perspective has been applied. In some embodiments, the event distributor may batch and transmit the events to the end recipient after the even priority has been determined. In one implementation, a syslog protocol may be used to send each event as a line of log data to be processed by an end consumer—usually a display/query engine. The transmission of events requires an additional layer of asynchronous processing as the response time from the recipient cannot be allowed to slow down the higher level global perspective and contextual analysis work which is ongoing in other threads. Therefore, a transmission engine of the event distributor may build the final event line, packages it together with information about the host itself, and then send it directly into a separate queue where it will then be sent shortly after (usually a few seconds) while maintaining the pre-set timestamp of precisely when the event took place even if it is sent several seconds later.

The recipient of the event, whether hosted on a third party platform or using other proprietary software, may then display single events and/or perform wider analysis on groups of events, producing dashboard views and graphs to aid the administrator in identifying events that are most pertinent to their current investigation.

In accordance with at least one embodiment of the present disclosure, a method is provided, the method comprising gathering one or more events defining an action of a first object acting on a target; generating a contextual state for at least one of the one or more events by correlating the at least one event to an originating object, the contextual state including an indication of the originating object of the first object and an indication of at least one of a device on which the first object is executed and a user associated with the first object; obtaining a global perspective for the at least one event by obtaining information associated with one or more of the first object and the originating object, the information including at least one of age, popularity, a determination as to whether the first object is malware, a determination as to whether the originating object is malware; and Internet Protocol (IP) Address or Uniform Resource Locator (URL) information; assembling an event line including details associated with the at least one event, the details including information uniquely identifying each of the first object, the action of the first object, the target, and the originating object; and transmitting the assembled event line.

In accordance with further embodiments of the present disclosure, a system is provided, the system comprising a

device including a communication interface; a processor; data storage; and a sensor agent stored on the data storage that is executable by the processor, wherein the sensor agent is operable to: gather one or more events defining an action of a first object acting on a target; generate a contextual state for at least one of the one or more events by correlating the at least one event to an originating object, the contextual state including an indication of the originating object of the first object and an indication of at least one of a device on which the first object is executed and a user associated with the first object; obtain a global perspective for the at least one event by obtaining information associated with one or more of the first object and the originating object, the information including at least one of age, popularity, a determination as to whether the first object is malware, a determination as to whether the originating object is malware; and Internet Protocol (IP) Address or Uniform Resource Locator (URL) information; assemble an event line including details associated with the at least one event, the details including information uniquely identifying each of the first object, the action of the first object, the target, and the originating object; and transmit the assembled event line utilizing the communication interface.

In accordance with further embodiments of the present disclosure, a system is provided, the system comprising a device including a communication interface; a processor; data storage; and a sensor agent stored on the data storage that is executable by the processor, wherein the sensor agent is operable to: gather one or more events defining an action of a first object acting on a target; generate a contextual state for at least one of the one or more events by correlating the at least one event to an originating object, the contextual state including an indication of the originating object of the first object and an indication of at least one of a device on which the first object is executed and a user associated with the first object; assemble an event line including details associated with the at least one event, the details including information uniquely identifying each of the first object, the action of the first object, the target, and the originating object; and transmit the assembled event line utilizing the communication interface.

In accordance with at least one embodiment of the present disclosure, a method is provided, the method comprising obtaining an event line from one or more computing systems; parsing the event line to retrieve event information for at least one event, wherein the event information includes an event type, a contextual state for the event, a global perspective for the event, and a priority for the event; and displaying the event information for the at least one event to a display device.

In accordance with further embodiments of the present disclosure, a system is provided, the system comprising a device including a communication interface; a processor; data storage; a display device; and a parser stored on the data storage that is executable by the processor, wherein the parser is operable to: obtain an event line from one or more computing systems utilizing the communication interface; parse the event line to retrieve event information for at least one event, wherein the event information includes an event type, a contextual state for the event, a global perspective for the event, and a priority for the event; wherein the processor is operable to cause the event information for the at least one event to be displayed at a display device.

Additional features and advantages of embodiments of the present invention will become more readily apparent from the following description, particularly when taken together with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates a comparative example of a communication network, edge device, and one or more computing devices;

FIG. **2** depicts a communication network including one or more computing systems, global perspective information server, and one or more end recipients in accordance with an exemplary embodiment of the present disclosure;

FIG. **3** illustrates a block diagram depicting details of a computing system and a bridge in accordance with an exemplary embodiment of the present disclosure;

FIG. **4** illustrates a block diagram depicting details of a global perspective information server and an end recipient in accordance with an exemplary embodiment of the present disclosure;

FIGS. **5**A-C illustrates exemplary events, events having an applied contextual state, and events having an applied global perspective in accordance with an exemplary embodiment of the present disclosure;

FIG. **6** illustrates a flow chart depicting details of at least one embodiment in accordance with an exemplary embodiment of the present disclosure;

FIG. **7** illustrates a flow chart depicting operational details of a system filter in accordance with an exemplary embodiment of the present disclosure;

FIG. **8** illustrates a flow chart depicting operational details of a local aggregator and interpreter in accordance with an exemplary embodiment of the present disclosure;

FIG. **9** illustrates a flow chart depicting operational details of a context analyzer in accordance with an exemplary embodiment of the present disclosure;

FIG. **10** illustrates a flow chart depicting operational details of a global perspective unit and/or global perspective module in accordance with an exemplary embodiment of the present disclosure;

FIG. **11** illustrates a flow chart depicting operational details of an event priority generation unit in accordance with an exemplary embodiment of the present disclosure;

FIG. **12** depicts a first data structure employed in accordance with at least some embodiments of the present disclosure; and

FIG. **13** depicts one or more dashboards in accordance with at least some embodiments of the present disclosure.

DETAILED DESCRIPTION

FIG. **1** depicts aspects of a system **200** for providing forensic visibility into one or more computing devices **204**, **208**, and **212** in accordance with at least one embodiment of the present disclosure. In general, the system **200** includes one or more computing device **204**, each of which are interconnected to a global perspective information server **216** and end recipient **220** by one or more communication networks **212**. The computing device **204** may comprise a general purpose computer, such as but not limited to a laptop or desktop personal computer **204**B, a tablet computer, a smart phone **204**A, or other device capable of communications over a communication network **212** and capable of presenting content to an associated user. The computing device **204** may also comprise a processing unit that is interconnected to an affiliated input/output device.

The communication network **212** may include one or more networks capable of supporting communications between nodes of the system **200**, including but not limited to a computing devices **204**A-C, the global perspective information server **216** and the end recipient **220**. Examples

of communication networks **212** include the Internet or any wide area network (WAN), local area network (LAN), or networks in various combinations. Other examples of communication networks **212** in accordance with embodiments of the present disclosure include wireless networks, cable networks, satellite networks, and digital subscriber line (DSL) networks. In addition, different communication networks **212** may share a common physical infrastructure, for at least some portion of the physical network. For instance, a computing device **204** may be interconnected to a communication network **212** comprising the Internet and to a separate communication network between the computing device **204** and an end recipient **220**.

The computing device **204** includes and/or executes a sensor agent **208** in connection with the presentation of content to the user. In accordance with at least one embodiment of the present disclosure, six components, or modules, may be utilized by the sensor agent **208** to gather and process events. The sensor agent **208** may include, but is not limited to, low level system filters, a local aggregator and interpreter, a context analyzer, a global perspective module, an event priority module, and an event distributor. Of course, and as can be appreciated by those skilled in the art, all six modules may not be required by the sensor agent **208**. In at least one embodiment, the sensor agent may comprise fewer than six modules. That is, at least some functionality of the one or more modules may be provided by another device, such as a global perspective information server **216** for example. Moreover, it should be understood that the sensor agent **208** may include varying combination of modules. For example, the sensor agent **208** may include two of the six modules, three of the six modules, four of the six modules, or five of the six modules.

The system filters may intercept system events in a manner such that they do not impact system performance. Events may define those actions or behaviors of an object acting upon another object or some other entity. An object in this sense may be a computer file, part of a file or a sub-program, macro, web page or any other piece of code to be operated by or on the computer, or any other event whether executed, emulated, simulated or interpreted. Events may include, but are not limited to file activity, registry events, code events, network/internet events, process/thread events, window/GDI events, input events to name a few. An event may have three principal components: information and/or details describing the object performing the act (the "Actor"), information and/or details describing the act being performed (the "Event Type"), and information and/or details describing the object or identity of another entity upon which the act is being performed (the "Victim" or "Target"). For example, the event data might capture the identity, e.g. the IP address or URL, of a network entity with which an object is communicating, another program acting on the object or being acted on by the object, a database or IP registry entry being written to by the object, etc. While simple, this structure allows a limitless series of behaviors and relationships to be defined. Examples of the three components of an event might include, but are not limited to, those illustrated in Table 1.

TABLE 1

Actor	Event Type	Victim
Object 1	Creates Program	Object 2
Object 1	Sends data	IP Address 3
Object 1	Deletes Program	Object 4

TABLE 1-continued

Actor	Event Type	Victim
Object 1	Executes	Object 2
Object 2	Creates registry key	Object 4

Moreover, in some embodiments, the Actor, Event Type, and/or Victim may be represented in the form of a so-called signature or key relating to the object key. Key generation is mainly to keep the data storage and transmission requirements as minimal as possible. The keys for each object may be formed by a hashing function operating on the object at the computing system **204**. Accordingly, an event may include the three principal components as a key of the object performing the act (the "Actor"), a key of the act being performed (the "Event Type"), and the key of the object or identity of another entity upon which the act is being performed (the "Victim" or "Target"). Additionally, events at this level tend to be very raw and generally not individually useful—on an average system, tens of millions of events take place every minute and the noise ratio can prevent forensic solutions from being able to provide sufficient value to the end consumer of their data due to the inability to quickly find important events. Accordingly, in some embodiments, events are gathered and time stamped accurately and sequentially, establishing a chronological order of system events for consumption at a higher level.

The second component, the local aggregator and interpreter, receives events from the low level system filters in batches from the queues in which they are inserted in real-time and transmits them to the local service process where they are held for further analysis. In accordance with at least some embodiments of the present disclosure, the events may be cached and normalized such that kernel filenames, user mode filenames, process objects, registry objects, and other system structures are converted into a standard format for ease of consumption by one or more automated parsers. Following the aggregation and interpretation of the events, the context analyzer creates, or obtains, a contextual state by correlating events to objects and creating an audit trail for each individual event. In some instances, the sensor agent **208** of the computing device **204** may determine if contextual state information is available locally within a cache. If the contextual state information is not available locally, the sensor agent **208** of the computing device may obtain such contextual state information from a source of contextual state information accessible to the sensor agent **208**.

Contextual state information may include, but is not limited to, originating processes, devices, and users. Additionally, contextual state information may include GenesisActor information and ancillary data. GenesisActor information may contain the object that is not the direct Actor of an event but which is the ultimate parent of the event being performed. For example in the case of a software installation, the GenesisActor would be the object that the user or system first executed and that initiated the software installation process, e.g. Setup.exe. Ancillary data may include data associated with the event type, actor and victim. For example, an event such as that used to record the creation of a registry run key may identify the Actor object as creating a registry run key, the event type itself (e.g. "regrunkey"), and the Victim or subject of the registry run key. The ancillary data in this case would define the run key entry itself; the Hive, Key name and Value. As a contextual state is established for each event, the contextual state may be

cached and aggregated to be quickly queried in subsequent lookups further reducing latency and improving the responsiveness of the computing device **204**. Contextual state information may be provided by the sensor agent **208** and/or a source of contextual state information accessible to the sensor agent **208**. For example, the global perspective information server **216**, in some embodiments, may be able to provide contextual state information to the sensor agent **208** for one or more events. Moreover, such contextual state information may be provided from one or more other computing devices **204**.

After the contextual analysis completes, the sensor agent **208** establishes a global perspective for each event by calling on the global perspective information server **224**. For example, the sensor agent **208** of the computing device **204** may determine if global perspective information is available locally within a cache. If the global perspective information is not available locally, the sensor agent **208** of the computing device **204** may issue a global perspective request **224** utilizing the communication network **212** and destined for the global perspective information server **216**. The global perspective information server **216** may respond with global prospective information **228** and such information may be provided to the sensor agent **208** associated with the computing device **204**.

The global perspective information **228** may include information related to the age, popularity, and determination of an object of the event—whether it is known good, bad, or unknown. The global perspective information may be retrieved or otherwise obtained for each object of an event and any objects provided as a part of a contextual information state. In some instances, the reputation and category of each website and IP address are obtained as they are contacted. Such information may then be normalized.

The event priority module may contain priority logic, or one or more rule sets, to associate a priority to one or more events. The event priority module may utilize information from the local aggregator and interpreter, the context analysis module, and the global perspective module, to determine, or otherwise assign, a priority to an event; such a priority may be based on a rule set. The rule set may be provided by an administration system, backend system, end recipient **220**, or other component of a forensic system. For example, a rule set may determine that an event meeting certain conditions be assigned a priority according to a scale of severity. For instance, a rule set may specify that an event having a specific actor, victim, and/or origination process together with a global perspective indicating a specified age, popularity, malware determination, and/or IP/URL reputation be assigned an event priority representative of a potential risk assigned to the computing device **204** and/or network. Accordingly, when the event line containing event priority information is received at an end recipient **220**, the end recipient **220** may be able to more easily identify events, and/or instances of malware, that require more immediate attention.

An event distributor of the sensor agent **208** may then create an event line comprising information from the event, information from the contextual state, information from the global perspective, and information from regarding the priority of the event. Alternatively, or in addition, an event distributor of the global perspective information server **216** may create an event line comprising information from the event, information from the contextual state, and information from the global perspective.

Accordingly, once the event distributor of the sensor agent **208** and/or the event distributor of the global perspective information server **216** create the event line for at least one event, the event line **244**A-B may be sent to an end recipient **220** from the computing device **204** and/or the global perspective information server **216**. The event distributor may batch and transmit the events to the end recipient **220** after the global perspective has been applied. In at least one implementation, the syslog protocol is utilized to send each event as a line of log data to be processed by an end consume such as a display/query engine like the product "splunk".

Alternatively, or in addition, some embodiments of the present disclosure may feature an optional bridge **222** which may be a dedicated device residing locally on a network. The bridge **222** may aggregate communications to and from one or more computing system **204** such that information to and from sensor agent **208** is cached at the bridge **222**, optimizing global perspective requests and other communications. That is, the bridge may be shared across some or all devices in a network. In some embodiments, the bridge **222** may act as a local cache or data store for caching requests to the global perspective information server **216** and event lines directed to the end recipient **220**. Additionally, the bridge **222** may perform selective or intelligent caching of global perspective information such that subsequent lookups or requests of global perspectives may be served from the local cache of the bridge **222**.

FIG. **3** illustrates additional aspects of a system **200** in accordance with embodiments of the present invention, and in particular illustrates additional features and components of a computing device **204** and a bridge **222**. The computing device **204** may comprise a general purpose computer, tablet computer, smart phone, or other device or federated group of devices capable of supporting communications over a communication network **212**. The computing device **204** may include a processor **316**, memory **304**, one or more user input devices **308**, such as a keyboard and a pointing device, and one or more user output devices **312**, such as a display, speaker, and/or printer. Alternatively, or in addition, the user input **308** and the user output **312** may be combined into one device, such as a touch screen display. Computing device **204** may further include a communication interface **328** for communicating with another computing device **204**, a global perspective information server **216**, an end recipient **220**, and/or the communication network **212**.

Processor **316** is provided to execute instructions contained within memory **304** and/or storage **320**. As such, the functionality of the computing device is typically stored in memory **304** and/or storage **320** in the form of instructions and carried out by the processor **304** executing such instructions. Accordingly, the processor **316** may be implemented as any suitable type of microprocessor or similar type of processing chip. One example of the processor **304** may include any general-purpose programmable processor, digital signal processor (DSP) or controller for executing application programming contained within memory **304** and/or storage **320**. Alternatively, or in addition, the processor **316**, memory **304**, and/or storage **320**, may be replaced or augmented with an application specific integrated circuit (ASIC), a programmable logic device (PLD), or a field programmable gate array (FPGA).

The memory **304** generally comprises software routines facilitating, in operation, pre-determined functionality of the computing device **204**. The memory **304** may be implemented using various types of electronic memory generally including at least one array of non-volatile memory cells (e.g., Erasable Programmable Read Only Memory (EPROM) cells or FLASH memory cells, etc.). The memory **304** may also include at least one array of dynamic random

access memory (DRAM) cells. The content of the DRAM cells may be pre-programmed and write-protected thereafter, whereas other portions of the memory may selectively be modified or erased. The memory **304** may be used for either permanent data storage and/or temporary data storage.

The data storage **320** may generally include storage for programs and data. For example, data storage **320** may provide storage for a sensor agent **208** including the system filters **340**, the local aggregator and interpreter **344**, the context analyzer **348**, the global perspective module **352**, the event distributor **356**, and/or the general operating system and other programs and data **324**. Additionally, the data storage **320** may also include an event store **328**, such as a database, for storing and/or caching event information, context information, global perspective information, and/or event lines. One or more components of the computing device **204** may communicate with one another utilizing a bus **336**. Alternatively, or in addition, the sensor agent **208** may be provided separate and apart from data storage **320**. That is, the sensor agent **208**, including the system filters **340**, the local aggregator and interpreter **344**, the context analyzer **348**, the global perspective module **352**, the event priority generation unit **358**, and the event distributor **356** may be implemented using a processor, such as a microprocessor **304**. Accordingly, the microprocessor **304** may be specifically programmed to carry out one or more functions of the sensor agent **208**. For example, the processor **304** may execute instructions associated with the sensor agent **208**, including the system filters **340**, the local aggregator and interpreter **344**, the context analyzer **348**, the global perspective module **352**, the event priority generation unit **358**, and the event distributor contained within memory **304** and/or storage **320**. Alternatively, or in addition, the sensor agent **208**, including the system filters **340**, the local aggregator and interpreter **344**, the context analyzer **348**, the global perspective module **352**, the event priority generation unit **358**, and the event distributor **356** may share the processor **304**.

The system filters **340** intercept system events in a manner such that they do not impact system performance. In some embodiments, events are gathered and time stamped accurately and sequentially such that a chronological order of system events is maintained and may be provided for consumption at a higher level. As one example, the time stamp may correspond to the time at which the event took place. Such a timestamp may be added to the event and/or maintained at an event line. Moreover, the system events may then be sorted according to event type and/or processing resources such that the system events are inserted, or stored, in a queue in real-time.

The local aggregator and interpreter receives events from the system filters **340**, and collects the received events into batches from the queues in which they are inserted in real-time. The batches of events are then transmitted to a local service process where the events are held for further analysis. Such events may be held, or otherwise stored, within an event store **328**. At this stage, intelligent caching may be invoked to drop events which are duplicates of recent events, and some data normalization may be performed to ensure consistency. Caching may be implemented sparingly but in critical areas to ensure redundant events are not continually reported—many applications perform thousands of the same events (network access to the same destination IP address, for example) and transmitting these events into an external log source would create a significant amount of unnecessary noise and potential performance degradation. The cache may integrate a "time to live" concept to prevent applications from hiding their traffic within their noise. For example, events meeting predefined criteria may be dropped from the real-time system event queues. In some embodiments, if the predefined criteria include determining if a system event is the same or otherwise matches a previous system event, then the event may be dropped. Additionally, if a duplicate event occurs within a specified time, the event may be dropped from the queue. Alternatively, or in addition, if the duplicate system event occurs within a specified period of time, the duplicate system event may be modified to reflect that the duplicate system event is the same as a previously occurring system event. Accordingly, the later occurring duplicate system event may receive all contextual state information and global perspective information without requiring all of the determinations and/or lookups of such information. However, a time-to-live (TTL) functionality may be implemented on a cache basis to ensure that cached events, or previous events having context and/or a global perspective, are current. Accordingly, not all previous events having contextual information may be available to provide contextual information to a current event. As another option, the later occurring duplicate system event may be modified in a manner such that when the later occurring duplicate system event is interpreted by the end recipient **220**, the end recipient **220** may determine that the system event information may be the same as a previously parsed system event line. Accordingly, a full event line for the duplicate system event is not generated and/or transmitted saving bandwidth and information processing resources. Further, the caches may automatically roll over to ensure sufficient space is always available without requiring exorbitant amounts of memory.

The local aggregator and interpreter **344** may additionally perform data normalization on items such as kernel filenames, user mode filenames, process objects, registry objects, and other system structures and normalizes them into a standard format which is easy more easily consumed by automated parsers. Consolidating and normalizing data at this stage allows the data to be handled much more quickly at the higher levels to avoid reprocessing data and potentially missing events just because they were referred to slightly differently (\Device\HarddiskVolume1\Windows\file.exe instead of C:\Windows\file.exe for example). In some embodiments, an aspect of this subsystem provides the ability to process events asynchronously. The subsequent modules which will execute over the data may potentially take a considerable amount of time to complete but delaying the response for each event at this phase would result in a significant system slowdown. Instead, events are intelligently queued in multiple layers to minimize the number of individual threads but still ensure each event is given sufficient time to be independently processed. Because of this structure, there is little to no performance difference if an event takes one millisecond to be processed or one hour for the full series of routines to complete.

The context analyzer **348** ensures that a user at the end recipient **220** is provided with the deepest view of the most important data but still has access to the full range of data if they wish to see it. Accordingly, the context analyzer **348** correlates objects within system events to additional objects known to exist. For example, the context analyzer **348** may correlate an actor, victim, and/or event type to one or more originating processes, devices, and users. Such a correlation creates an audit trail for each individual event. As will be described later, this audit trail allows an administrator, at an

end recipient **220** for example, to forensically track back any event which occurs to what triggered it. Accordingly, this additional information, including the originating processes, devices, and user, allows a contextual state to be established for each event. As each contextual state is established, the contextual state may be cached and aggregated to be quickly queried in subsequent lookups further reducing latency and improving the performance of the system.

After the context analyzer completes, the sensor agent **208** may attempt to establish a global perspective for each event by calling the cloud database utilizing an infrastructure similar to the infrastructure discussed in U.S. application Ser. No. 13/372,375 filed Feb. 13, 2012, "Methods and Apparatus for Dealing with Malware" (US-2012-0260340-A1, but with an additional layer capable of responding in a highly scalable manner which avoids the overhead generally associated with the complex rule processing within the overall database. Such an additional layer may be named SkyMD5. The response from the SkyMD5 layer may include information related to the age, popularity, and determination of an object—whether it is known good, bad, or unknown. This data may then be normalized and appended to each relevant portion of the event line. Further lookups are performed for network activity, assessing the reputation and category of each website and IP address as they are contacted.

In addition, similar to the contextual analyzer, the global perspective unit may first determine whether or not global perspective information for the particular system event is available locally. For example, global perspective information may be cached or otherwise stored in the storage data **320**. In addition, a TTL approach may be additionally utilized to reduce the amount of traffic, or calls, to the global perspective information server **216**, by the global perspective unit **352** of the sensor agent **208**. Accordingly, contextual state information may only be available locally for a specified period of time and/or specified number of system events.

The event priority module **358** may contain priority logic, or one or more rule sets, to associate a priority to one or more events. The event priority module may utilize information from the local aggregator and interpreter **344**, the context analysis module **348**, and the global perspective module **352**, to determine, or otherwise assign, a priority to an event; such a priority may be based on a rule set. The rule set may be provided by an administration system, backend system, end recipient **220**, or other component of a forensic system. For example, a rule set may determine that an event meeting certain conditions be assigned a priority according to a scale of severity. Such an event priority may then be added to an existing event line or data structure.

The event distributor **356** may batch and transmit the events to the end recipient **220** after the global perspective has been applied. Alternatively, or in addition, the event distributor **356** may batch and transmit the events to the end recipient **220** after an event priority has been determined for the event. In one implementation, the syslog protocol is used to send each event as a line of log data to be processed by an end recipient **220**. The transmission of events requires an additional layer of asynchronous processing as the response time from the recipient cannot be allowed to slow down the higher level global perspective and contextual analysis work which is ongoing in other threads. Therefore, a transmission engine of the event distribute **356** builds the final event line, packages the event line together with information about the host itself, and then sends it directly into a separate queue where it will then be sent shortly after (usually a few seconds) while maintaining the pre-set timestamp of precisely when the event took place even if it is sent several seconds later. The separate queue may be a queue located in the storage data **320** and more specifically in the event store **328**.

The bridge **222** may comprise a general purpose computer and/or server or other device or federated group of devices capable of supporting communications over a communication network **212**. The bridge **222** may include a processor **382**, memory **362**, one or more user input devices **366**, such as a keyboard and a pointing device, and one or more user output devices **378**, such as a display, speaker, and/or printer. The bridge **222** may further include a communication interface **386** for communicating with another computing device **204**, an end recipient **220**, a global perspective information server **216**, and/or the communication network **212**. Processor **382** and memory **362** may be the same or similar as that which was described with respect to processor **316** and memory **304** respectively.

The data storage **370** may generally include storage for programs and data. For example, data storage **370** may provide storage for a cached event store **394** and/or the general operating system and other programs and data **374**. Optionally, as indicated by the dashed line, the cached event store **394** may be separate and apart from the data storage **370**. That is, the cached event store **394** functions independently and/or in cooperation with memory **362** and processor **382**.

The bridge **222** may be a dedicated device residing locally on a network. The bridge **222** may aggregate communications to and from one or more computing system **204** such that information to and from sensor agent **208** is cached and stored at the cached event store **394**, optimizing global perspective requests and other communications. Additionally, the bridge **222** may perform selective or intelligent caching of global perspective information such that subsequent lookups or requests of global perspectives may be served from the cached event data store **394**. One or more components of the bridge **222** may communicate with one another utilizing a bus **390**.

FIG. **4** illustrates additional aspects of a system **200** in accordance with embodiments of the present invention, and in particular illustrates additional features and components of a global perspective information server **216** and an end recipient **220**. The global perspective information server **216** may comprise a general purpose computer and/or server or other device or federated group of devices capable of supporting communications over a communication network **212**. The global perspective information server **216** may include a processor **464**, memory **452**, one or more user input devices **456**, such as a keyboard and a pointing device, and one or more user output devices **460**, such as a display, speaker, and/or printer. The global perspective information server **216** may further include a communication interface **468** for communicating with another computing device **204**, an end recipient **220**, and/or the communication network **212**. Processor **464** and memory **452** may be the same or similar as that which was described with respect to processor **316** and memory **304** respectively.

The data storage **488** may generally include storage for programs and data. For example, data storage **488** may provide storage for a global perspective module **476**, data store **486**, and/or the general operating system and other programs and data **484**. Optionally, each of the global perspective module **476**, the data store **486**, and/or the event distributor **492** may be separate and apart from the data storage **488**. That is, as indicated by the dashed line, each of

the global perspective module **476**, the data store **486**, and/or the event distributor **492** functions independently and/or in cooperation with memory **452** and processor **464**.

The global perspective module **476** provides global perspective information at the request of a sensor agent **208**. For example, and as previously described, the sensor agent **208** may transmit a global perspective request **224** to the global perspective information server **216** requesting global perspective information for one or more system events having a contextual state. Accordingly, the global perspective module **476** may provide the global prospective information **228** to the global perspective unit **352** of the computing system **204**. In addition, in some embodiments the event distributor functionality may reside at the global perspective information server **216**. Accordingly, in such an embodiment, the global perspective module **476** may receive event information having contextual state information. In such an instance, the global perspective module **476** may provide the global perspective information and store the system event, contextual information, and global perspective information in data store **486**. The event distributor **492** may then assemble the event line information, queue the event line for transmission, and transmit the event line to the end recipient **220**. The event distributor **492** may function and/or operate in a manner similar to or the same as the event distributor **356** of the sensor agent **208**. One or more components of the global perspective information server **216** may communicate with one another utilizing a bus **472**.

The end recipient **220** may comprise a general purpose computer and/or server or other device or federated group of devices capable of supporting communications over a communication network **212**. The end recipient **220** may include a processor **416**, memory **404**, one or more user input devices **408**, such as a keyboard and a pointing device, and one or more user output devices **412**, such as a display, speaker, and/or printer. The end recipient **220** may further include a communication interface **428** for communicating with another computing device **204**, a global perspective information server **216**, and/or the communication network **212**. Processor **416** and memory **404** may be the same or similar as that which was described with respect to processor **316** and memory **304** respectively.

The end recipient **220** may further include data storage **420** which may include storage for programs and data. For example, data storage **420** may provide storage for an event line parser **440**, a user interface generator **444**, a global information store **448**, and/or the general operating system and other programs and data **420**. Optionally, as indicated by the dashed line, each of the event line parser **440**, the user interface generator **444**, and the global information store **448** may be provided separate and apart from the data storage **436** where each of the event line parser **440**, a user interface generator **444**, and the global information store **448** functions independently and/or in cooperation with memory **404** and processor **416**. One or more components of the end recipient **220** may communicate with one another utilizing a bus **432**.

The event line parser **440** may parse an incoming event line **244**A-B and store the event lines, either parsed or whole, at the global information store **448**. Accordingly, the end recipient **220** may utilize the parsed event line to generate an user interface utilizing the user interface generator **444**. The generated user interface may then be displayed to a user at an user output device **412**. The user interface generator **444** may create one or more dashboards that provide information to a user of the end recipient **220**. Accordingly, dashboards may display correlated events collected by a sensor agent **204**. Such correlated events may comprise File System/Disk activity, network activity, and registry activity on the endpoint along with the global community data and IP and domain reputation data to provide a complete point of view of the computing system **204** activity and further provide a full meaning to every single reported event. Accordingly, spotting and highlighting behaviours in the event data from computing systems **204** may potentially assist in finding and determining whether such activity is malicious or not.

Moreover, there may exist multiple dashboards each of which provide a specific heuristic that targets a specific potential malicious behaviour. For example, a first dashboard generally looks for executables that have been found and executed from specific critical paths in the computing system **204**; those paths that are known to host malware executables (browser's cache, temporary files folder, system32 folder, auto start folder and so on). The dashboard may filter those files determined as good, unknown and bad from global community information. For unknown files, the dashboard may show their popularity at the global community, i.e. on how many computing systems **204** they have been seen—and if they establish outbound internet connections. If such files connect to the Internet, the dashboard may trace down and identify the specific connection and geolocate the destination address on a world map.

In another embodiment, a separate dashboard may analyze outbound connections from the analyzed endpoints. Accordingly, this dashboard may filter out and highlight connections to high-risk countries, i.e. those countries known to host malicious servers, and retrieve software that actually established the suspicious connections. The dashboard may also provide another point of view; that is, the dashboard may highlight all connections to specific IPs and domains that have been set up recently (e.g. in the last 60 days). All connections may be shown on a map view and the contacted IPs and domains are crosschecked with a database to determine their reputation and website category (if known).

As another example, a dashboard may highlight potential exploits detected on the endpoint by monitoring the most common software usually installed on an average computing system **204** and targeted by exploit attacks. The dashboard may generally identify software like Java, Adobe Acrobat, Microsoft Office, Flash Player and monitor if such software is dropping executable files on the system. Accordingly, if the software behaves in this manner, such a behaviour should be considered a suspicious behaviour, as software should not be dropping any executable files on the system. If such an occurrence is detected, the dashboard may analyze the dropped executable for its popularity in the global community and then identify potential outbound connections established by the just-dropped executable. If an outbound connection is found, the dashboard retrieves the destination IP and renders, or draws, the location of the IP address on a map view, such as a world map.

As another example, a separate dashboard may highlight those new processes created on the monitored computing systems **204** that are undetermined and have only been recently seen in the global community. For every process, a lifetime scheme is assembled which allows the administrator to understand when the undetermined file firstly landed on the endpoint and for how long time it has been active on the system. This dashboard may further provide specific events triggered by the suspicious process while active on the system.

As another example, a dashboard may highlight system processes that have been found to establish concurrent connections to different IPs in less than a specific timeframe and/or threshold, such as one second. If found, the system processes may be suspect and deemed to host malicious code; such malicious code having been injected into the website. For every established connection, the dashboard may trace and identify the contacted country and the specific IP/domain. As another example, a dashboard may be generated by the user interface generator **448** such the dashboard monitors activities on remote storage services like Dropbox, showing the filename and size of the files being saved on a remote server. The dashboard may further monitor and identify sensitive or classified company data that may be in the process of being stolen or placed (i.e. saved, stored, etc.) on a non-approved storage location.

FIGS. **5**A-**5**C illustrate additional aspects of a system **200** in accordance with embodiments of the present invention, and in particular illustrates event capturing, the application of contextual state, and the application of a global perspective to such captured events. For example, events **508**A-**520**A are representative events that may have been captured by one or more system filters **340** of a sensor agent **204**. The system events **504**A-**520**A may include an Event ID, an Event Type, an Actor and a Victim. Although not illustrated, a timestamp may also be included in the event. As previously described, the actor and the victim may comprise objects. For example, in the system event **520**A, an event type of Registry Edit is captured where Object 2 is modifying Object 3; accordingly, Object 3 may be the registry hive, a specific registry entry, and/or a specified registry value. Moreover each of the Event ID, Event Type, Actor, and Victim may be hashed resulting in a key that uniquely identifies the data.

As illustrated in FIG. **5**B, a contextual analysis may be applied to the captured events **504**B-**520**B. Accordingly, a context **520**B comprising an origination proceess, Device, User, and in some instances, a GenesisActor may further supplement or enhance the Event ID, Event Type, Actor, and Victim. Although illustrated as including an originating process, GenesisActor, device, and user, the contextual information forming the contextual state **520**B may include more or less information and in varying combinations. Moreover, as illustrated in FIG. **5**B, the originating process or object may determined based on a previous event that initiated or otherwise created the current process and/or object. Additionally, as illustrated in FIG. **5**B, the GenesisActor is determined based on a chain of previous events. For example, the Actor of Object 1 for system event **504**B of Event Type "executes" operates on Filename 1. Object 1 subsequently creates a process Object 2 in **508**B which is further utilized in both **516**B and **520**B. Accordingly, as the Object 1 initiated subsequent events, Object 1 may be determined to be the GenesisActor for **520**B, **516**B, **512**B, and **508**B.

As illustrated in FIG. **5**C, a global perspective is applied to event **520**C. The global perspective may comprise information for each object in the event and/or the contextual state **524**. For example, a global perspective may be obtained for Object 2, Object 3, and Object 1 as illustrated. Accordingly, a global perspective **528** indicating the age of Object 2, the popularity of Object 2, the determination of Object 2, and/or additional IP/URL information of Object 2 may be provided. Similar global perspective information **532**, **536** may be obtained for Object 3 and Object 1 respectfully. Accordingly, an event line may be assembled utilizing the event information, contextual state, and global perspective information. In addition, the global perspective may provide additional information pertaining to an IP/URL address. For example, such information may include a categoryID of the IP/URL, a confidence of such a categoryID, a reputation, geolocation information such as which country the IP/URL is associated with, and/or a threat history associated with the IP/URL.

Referring now to FIG. **6**, a method **600** for providing forensic visibility into a system is provided in accordance with embodiments of the present disclosure. Method **600** is in embodiments, performed by a device, such as the sensor agent **204** and/or the end recipient **220**. More specifically, one or more hardware and software components may be involved in performing method **600**. In one embodiment, one or more of the previously described units perform one or more of the steps of method **600**. The method **600** may be executed as a set of computer-executable instructions executed by a computer system and encoded or stored on a computer-readable medium. Hereinafter, the method **600** shall be explained with reference to systems, components, units, software, etc. described with FIGS. **1**-**5**C.

Method **600** may continuously flow in a loop, flow according to a timed event, or flow according to a change in an operating or status parameter. Method **600** is initiated at step **604** where events of a computing system **204** may be initiated. At step **608**, such an event may be detected. As previously discussed, the events, which may comprise system events, may be detected by one or more system filters **340** of a sensor agent **208**. The sensor agent **208** may be comparatively small compared with other commercially available forensic tools and anti-malware software. This may be achieved by the sensor agent **204** being developed in a low-level language, having direct access to system resources such as the video display, disk, memory, the network(s), and without the incorporation of many standard code or dynamic linked libraries to perform these functions. Memory usage may be optimized by storing data in a local database structure that provides the ability to refer to objects by a unique identifier rather than requiring full filenames or signatures. All unnecessary dynamic link libraries are unloaded from the process immediately as they are identified as no longer being used, and background threads are merged to reduce CPU usage. A small, efficient sensor agent **204** may be deployed more quickly and may be used alongside other programs, including other security programs, with less load on or impact on the computer's performance. This approach also has the advantage of having less surface area for attack by malware making it inherently more secure.

Method **600** then proceeds to step **612** where the local aggregator and interpreter **344** receives events from the low level system filters **340**, collecting them in batches from the queues in which they are inserted in real-time. As previously discussed, intelligent caching may be invoked to drop events which are duplicates of recent events. Further, some data normalization may be performed at step **612** to ensure consistency. Further still, caching may be implemented sparingly but in critical areas to ensure redundant events are not continually reported. Method **600** then proceeds to step **616** where a contextual state may be applied by a context analyzer **348** to the event information. Accordingly, the context analyzer correlates events to objects such that an audit trail for each individual event is created. As a contextual state is established, it is cached and aggregated to be quickly queried in subsequent lookups further reducing latency and improving the responsiveness of the system. Once a contextual state is applied, method **600** then proceeds to step **620** where a global perspective is applied for each

Case 6:22-cv-00243-ADA-DTG Document 69-1 Filed 10/14/22 Page 426 of 584

event and/or each object of the event and contextual state. Accordingly, information related to the age, popularity, and determination of an object—whether it is known good, bad, or unknown is obtained. In addition, as previously mentioned, the global perspective may contain information pertaining specifically to an IP/URL address. For example, the global perspective may include information relating network activity and an assessment of the reputation and category of each website and IP address as they are contacted. This data may then be normalized. Method **600** then proceeds to step **622** where an event priority may be applied to the event. As previously discussed, the event priority may be obtained utilizing a rule set and applying such a rule set to aggregated event information, contextual state information, and global perspective information. Accordingly, method **600** then proceeds to step **624** where such data is appended to each relevant portion of an event line.

Method **600** may then proceed to step **628** where the event distributor **356** may transmit the event line **244** to the end recipient **220**. The event distributor may first assemble the information and transmit the batched information after the global perspective has been applied. Once at the end recipient, the method **600** may then parse the event line at step **632** to filter out one or more events. Accordingly, the events may be stored in the data store **480**. Method **600** may then display the parsed event line at one or more dashboards in accordance with step **636**. Method **600** then ends at step **640**.

Referring now to FIG. **7**, additional details regarding step **608** of method **600** is provided in method **700**. Method **700** is in embodiments, performed by a device, such as the sensor agent **204** and/or the system filter **340**. More specifically, one or more hardware and software components may be involved in performing method **700**. In one embodiment, one or more of the previously described units perform one or more of the steps of method **700**. The method **700** may be executed as a set of computer-executable instructions executed by a computer system and encoded or stored on a computer-readable medium. Hereinafter, the method **700** shall be explained with reference to systems, components, units, software, etc. described with FIGS. **1-6**.

Method **700** may continuously flow in a loop, flow according to a timed event, or flow according to a change in an operating or status parameter. Method **700** is initiated at step **704** where events of a computing system **204** may be initiated. At step **708**, such an event may be detected. As previously discussed, the events, which may comprise system events, may be detected by one or more system filters **340** of a sensor agent **208**. Accordingly, at step **712**, an event record may be created. The event record may be the same or similar to that which was described with respect to FIG. **5**A. Accordingly, the event record may include an Event ID, and Event Type, an Actor, and a Victim. Moreover, in some embodiments, instead of creating a separate event record and a separate event line, a single event line for each event may be utilized. That is, the event record may be the event line. Accordingly, at step **716** the event line and/or the event record may be time stamped and stored. At step **720**, method **700** may then determine which queue the event record or event line, should be assigned. In some embodiments, the event record/event line may be assigned to a specified queue based on the event type. Alternatively, or in addition, the event record/event line may be assigned to a specified queue based on the needed resources and the available resources. Method **700** then ends at step **724**.

Referring now to FIG. **8**, additional details regarding step **612** of method **600** is provided in method **800**. Method **800** is in embodiments, performed by a device, such as the sensor agent **204** and/or the local aggregator and interpreter **344**. More specifically, one or more hardware and software components may be involved in performing method **800**. In one embodiment, one or more of the previously described units perform one or more of the steps of method **800**. The method **800** may be executed as a set of computer-executable instructions executed by a computer system and encoded or stored on a computer-readable medium. Hereinafter, the method **800** shall be explained with reference to systems, components, units, software, etc. described with FIGS. **1-7**.

Method **800** may continuously flow in a loop, flow according to a timed event, or flow according to a change in an operating or status parameter. Method **800** is initiated at step **804**. Method **800** then proceeds to step **808** where batches of event records/event lines are received form one or more previously described queues. For each event, method **800** proceeds to determine if the event record/event line meets a predefined criteria. For example, if the current event record/event line matches or is otherwise the same as a previous event record/event line, method **800** may proceed to step **824** where the current event record/event line may be dropped. Alternatively, or in addition, the dropped event record/event line may be modified such that either the contextual analyzer **348** and/or the global perspective unit **352** may be able to determine a contextual state and global perspective based on a previously captured event record/event line. Alternatively, or in addition, the dropped event record/event line may be modified to refer to a previously captured event record/event line such that upon parsing by the end recipient **220**, the current event line/event record is provided meaning from a previously parsed event record/event line

Accordingly, method **800** may then proceed to step **816** where the information may be normalized. The normalization may be implemented with a database which takes complete path names and translates them to representatives paths. So, for example, the Windows\file.exe would end up normalized to %windir%\file.exe, allowing rules to be created for any full path. Similar logic is utilized for paths like %appdata% which would have unique elements per-user and would be difficult to manage if not normalized. Accordingly, once the information in the event record/event line is normalized, method **800** may proceed to step **820** where the events may be intelligently queued. For instance, each event record/event line may be placed in a queue according to an event type. Method **800** may then end at step **828**.

Referring now to FIG. **9**, additional details regarding step **616** of method **600** is provided in method **900**. Method **900** is in embodiments, performed by a device, such as the sensor agent **204** and/or the context analyzer **348**. More specifically, one or more hardware and software components may be involved in performing method **900**. In one embodiment, one or more of the previously described units perform one or more of the steps of method **900**. The method **900** may be executed as a set of computer-executable instructions executed by a computer system and encoded or stored on a computer-readable medium. Hereinafter, the method **900** shall be explained with reference to systems, components, units, software, etc. described with FIGS. **1-8**.

Method **900** may continuously flow in a loop, flow according to a timed event, or flow according to a change in an operating or status parameter. Method **900** is initiated at step **904**. Method **900** then proceeds to step **908** where event records/event lines are received. Method **900** then proceeds to step **912** where objects of the event record/event line are correlated to objects in a contextual state. For example, and

as illustrated in at least FIG. **5**B, a contextual state may include originating processes, devices, and users. Accordingly, once objects of the event record/event line have been correlated, method **900** proceeds to step **916** where the event record/event line includes an established contextual state. In some embodiments consistent with the present disclosure, the contextual state may already be cached and may be obtained based on the event record/event line information. As one example, based on the event type, actor, and victim, it may be possible to recover a contextual state that includes an originating process, the device, and the user. However, not all cached instances of cached contextual state will suffice as different users may utilize a computing device **204**.

Method **900** then proceeds to step **920** where the contextual state may be cached and made accessible for later access. In accordance with at least one embodiment, the event line may be partially assembled where the partially assembled event line may include various combinations of the event type, the actor object, the victim object, and the origination process. Accordingly, the partially assembled event line may later be passed to a global perspective information server. As such, method **900** ends at step **928**.

Referring now to FIG. **10**, additional details regarding step **620** of method **600** is provided in method **1000**. Method **1000** is in embodiments, performed by a device, such as the sensor agent **204** and/or the global perspective unit **352**, the global perspective module **476**, and/or the global perspective information server **216**. More specifically, one or more hardware and software components may be involved in performing method **1000**. In one embodiment, one or more of the previously described units perform one or more of the steps of method **1000**. The method **1000** may be executed as a set of computer-executable instructions executed by a computer system and encoded or stored on a computer-readable medium. Hereinafter, the method **1000** shall be explained with reference to systems, components, units, software, etc. described with FIGS. **1-9**.

Method **1000** may continuously flow in a loop, flow according to a timed event, or flow according to a change in an operating or status parameter. Method **1000** is initiated at step **1004**. Method **1000** then proceeds to step **1008** where method **1000** determines whether event record/event line information is cached. Such a determination may be based upon whether the event information, such as the event type, the actor, and/or the victim, match a predetermined criteria. For instance, not all event types may be cached. In one embodiment, global perspective information caching will cache a specific number of event types occurring within a specified time period. In other instances, all global perspective information for all events may be cached. Accordingly, if it is determined that global perspective information is not cached, method **1000** proceeds to step **1016** where the global perspective information server may be contacted. Accordingly, such a request may provide the global perspective information server **216** with information in an event record and/or a partially assembled event line. Accordingly, the global perspective information or a particular event may be retrieved. The retrieved global perspective information may then be stored at step **1024**. For example, such information may be stored in event store **328**.

Otherwise, if global perspective information is cached for an event record/event line, the method **1000** may proceed to step **1012** were method **1000** may determine whether a time-to-live (TTL) has been exceeded. In some instances, the cached information may expire based on the frequency of occurrence for a particular event. For example, if an event occurs with a high frequency, the global perspective unit **352** may determine that for efficiency reasons, either resources and/or bandwidth, caching for a particular event would be beneficial to improving the overall efficiency of the sensor agent **208** and/or computing device **204**. Accordingly, the method **1000** then proceeds to step **1020** where the global perspective for the event is retrieved based on the cached entry. Otherwise, if the occurrence of the same event happens at a reduced frequency such that it would not be beneficial to cache global perspective information, it might be more beneficial to directly lookup and retrieve such information directly from the global perspective information server **216**. Accordingly, the method **1000** proceeds to step **1016** where global perspective information is received from the global perspective information server. In some instances, the cache of global perspective events may expire according to a predetermined time. At step **1024**, method **1000** may proceeds to step **1028** where method **1000** ends.

Referring now to FIG. **11**, additional details regarding step **624** of method **600** is provided in method **1100**. Method **1100** is in embodiments, performed by a device, such as the sensor agent **204**. More specifically, one or more hardware and software components may be involved in performing method **1100**. In one embodiment, one or more of the previously described units perform one or more of the steps of method **1100**. The method **1100** may be executed as a set of computer-executable instructions executed by a computer system and encoded or stored on a computer-readable medium. Hereinafter, the method **1100** shall be explained with reference to systems, components, units, software, etc. described with FIGS. **1-10**.

Method **1100** may continuously flow in a loop, flow according to a timed event, or flow according to a change in an operating or status parameter. Method **1100** is initiated at step **1104**. Method **1100** then proceeds to step **1108** where a priority rule set is retrieved. The priority rule set may be retrieved directly from an administration system and/or may be received locally from a storage location associated with the sensor agent **208** and/or the data storage **320**. The priority rule set may have one or more predefined criteria for assigning a priority level to an event. The priority level may represent a perceived severity or risk in which such an event poses. Method **1100** then proceeds to step **1112** where an event priority, or level, is assigned to the event.

In at least one embodiment, the priority level may be derived from aggregated and interpreted event information from the local aggregator and interpreter **344**, context analyzer **348**, and global perspective unit **352**. Accordingly, an event having a specified Actor, Victim, Target, originating process, or the like together with a global perspective indicating that at least one of the Actor, Victim, Target, origination process has been determined to be malware may result in a rule set that assigns a priority level to the event indicating that such the event should be immediately brought to the attention of an administrator. Accordingly, the event may be said to have a "high" priority or be a "high" priority event. Method **1100** may then end at step **1116**. With reference to FIG. **12**, details of a data structure **1200** will be described in accordance with embodiments of the present disclosure. In some embodiments, the data structure **1200** may be used as an event line and may contain details specific to an event, context, and global perspective. For example, an event line **1204** may generally include, without limitation, event information **1208**, a time stamp **1212**, context state information **126**, global perspective information **1220**, and a priority level **1222**, where the priority may be a value that ranges between a low and a high number. In some embodiments, the event field **1208** may further include Event ID

information **1224**, Event Type information **1228**, Actor **1232**, and Victim information **1236**. In some embodiments, the context state **1216** may include, without limitation, Originating Process **1240** information, Device information **1244**, User information **1248**, dependent data such as, but not limited to additional meta data and/or ancillary data describing or containing detailed information about one or more objects **1252**, and object info **1256**. In addition, the global perspective field may include, but is not limited to an age **1260**, popularity **1264**, malware determination **1268**, and/or IP/URL information **1272**, where IP/URL information may include, but is not limited to a categoryID of the IP/URL, a confidence of such a categoryID, a reputation, geolocation information such as which country the IP/URL is associated with, and/or a threat history associated with the IP/URL. Of course, additional relevant information may be included in the event line **1204** depending on the event.

FIG. **13** generally illustrates exemplary user interfaces, or dashboards **1304**, **1332**, and **1336**, which may be displayed at an end recipient **220** to a user at an user output device **412**. The user interface generator **444** may create the one or more dashboards **1304**, **1332**, and **1336** to provide information from an event to a user of the end recipient **220**. Accordingly, the dashboards **1304**, **1332**, and **1336** may display correlated events collected by a sensor agent **204**. Such correlated events may comprise File System/Disk activity **1308**, network activity **1312**, and registry activity **1316** on the endpoint along with the global community data and IP and domain reputation data **1324** to provide a complete point of view of the computing system **204** activity and further provide a full meaning to every single reported event, including at least one of an originating process, an Event Type, a Target, a Victim, and a Actor. Accordingly, spotting and highlighting behaviours in the event data from computing systems **204** may potentially assist in finding and determining whether such activity is malicious or not.

Moreover, there may exist multiple dashboards **1304**, **1332**, and **1336** each of which provide a specific heuristic that targets a specific potential malicious behaviour. For example, a first dashboard **1326** generally looks for executables that have been found and executed from specific critical paths in the computing system **204**; those paths that are known to host malware executables (browser's cache, temporary files folder, system32 folder, auto start folder and so on). The dashboard **1326** may filter those files determined as good, unknown and bad from global community information. For unknown files, the dashboard may show their popularity at the global community, i.e. on how many computing systems **204** they have been seen—and if they establish outbound internet connections. If such files connect to the Internet, the dashboard **1332** may trace down and identify the specific connection and geo-locate the destination address on a world map.

In another embodiment, a separate dashboard **1336** may analyze outbound connections from the analyzed endpoints. Accordingly, this dashboard may filter out and highlight connections to high-risk countries, i.e. those countries known to host malicious servers, and retrieve software that actually established the suspicious connections. The dashboard **1336** may also provide another point of view; that is, the dashboard **1336** may highlight all connections to specific IPs and domains that have been set up recently (e.g. in the last 60 days). All connections may be shown on a map view and the contacted IPs and domains are crosschecked with a database to determine their reputation and website category (if known).

As another example, a dashboard **1340** may highlight potential exploits detected on the endpoint by monitoring the most common software usually installed on an average computing system **204** and targeted by exploit attacks. The dashboard **1340** may generally identify software like Java, Adobe Acrobat, Microsoft Office, Flash Player and monitor if such software is dropping executable files on the system. Accordingly, if the software behaves in this manner, such a behaviour should be considered a suspicious behaviour, as software should not be dropping any executable files on the system. If such an occurrence is detected, the dashboard **1340** may analyze the dropped executable for its popularity in the global community and then identify potential outbound connections established by the just-dropped executable. If an outbound connection is found, the dashboard **1340** retrieves the destination IP and renders, or draws, the location of the IP address on a map view, such as a world map.

As another example, a separate dashboard **1344** may highlight those new processes created on the monitored computing systems **204** that are undetermined and have only been recently seen in the global community. For every process, a lifetime scheme is assembled which allows the administrator to understand when the undetermined file firstly landed on the endpoint and for how long time it has been active on the system. This dashboard **1344** may further provide specific events triggered by the suspicious process while active on the system.

As another example, a dashboard **1348** may highlight system processes that have been found to establish concurrent connections to different IPs in less than a specific timeframe and/or threshold, such as one second. If found, the system processes may be suspect and deemed to host malicious code; such malicious code having been injected into the website. For every established connection, the dashboard may trace and identify the contacted country and the specific IP/domain. As another example, a dashboard **1348** may be generated by the user interface generator **448** such the dashboard monitors activities on remote storage services like Dropbox, showing the filename and size of the files being saved on a remote server. The dashboard **1348** may further monitor and identify sensitive or classified company data that may be in the process of being stolen or placed (i.e. saved, stored, etc.) on a non-approved storage location.

In the foregoing description, for the purposes of illustration, methods were described in a particular order. It should be appreciated that in alternate embodiments, the methods may be performed in a different order than that described. It should also be appreciated that the methods described above may be performed by hardware components or may be embodied in sequences of machine-executable instructions, which may be used to cause a machine, such as a general-purpose or special-purpose processor or logic circuits programmed with the instructions to perform the methods. These machine-executable instructions may be stored on one or more machine readable mediums, such as CD-ROMs or other type of optical disks, floppy diskettes, ROMs, RAMs, EPROMs, EEPROMs, magnetic or optical cards, flash memory, or other types of machine-readable mediums suitable for storing electronic instructions. Alternatively, the methods may be performed by a combination of hardware and software.

Furthermore, embodiments may be implemented by hardware, software, firmware, middleware, microcode, hardware description languages, or any combination thereof. When implemented in software, firmware, middleware or micro-

code, the program code or code segments to perform the necessary tasks may be stored in a machine readable medium such as storage medium. A processor(s) may perform the necessary tasks. A code segment may represent a procedure, a function, a subprogram, a program, a routine, a subroutine, a module, a software package, a class, or any combination of instructions, data structures, or program statements. A code segment may be coupled to another code segment or a hardware circuit by passing and/or receiving information, data, arguments, parameters, or memory contents. Information, arguments, parameters, data, etc. may be passed, forwarded, or transmitted via any suitable means including memory sharing, message passing, token passing, network transmission, etc.

The foregoing discussion of the invention has been presented for purposes of illustration and description. Further, the description is not intended to limit the invention to the form disclosed herein. Consequently, variations and modifications commensurate with the above teachings, within the skill or knowledge of the relevant art, are within the scope of the present invention. The embodiments described hereinabove are further intended to explain the best mode presently known of practicing the invention and to enable others skilled in the art to utilize the invention in such or in other embodiments and with various modifications required by the particular application or use of the invention. It is intended that the appended claims be construed to include alternative embodiments to the extent permitted by the prior art.

What is claimed is:

1. A method comprising:

gathering an event defining an action of a first object acting on a target, wherein the first object is executed on a device;

generating contextual state information for the event by correlating the event to an originating object of the first object;

obtaining a global perspective for the event based on the contextual state information, wherein the global perspective comprises information associated with one or more of the first object and the originating object, and wherein the global perspective relates to one or more other events related to the event across a network;

generating an event line comprising information relating to the event, wherein the information relates to at least one of the first object, the action of the first object, the target, and the originating object; and

transmitting the generated event line.

2. The method of claim **1**, wherein the information associated with the one or more of the first object and the originating object is obtained from a community database; and

wherein the event line is transmitted to an end recipient.

3. The method of claim **1**, further comprising:

obtaining an event priority based on one or more of the generated contextual state information and the global perspective, wherein the event line further includes the event priority.

4. The method of claim **1**, wherein the event includes information identifying the first object performing the action, information identifying the action being performed, and information identifying the target upon which the act is being performed.

5. The method of claim **4**, further comprising:

determining whether the global perspective for the event has been cached;

when the global perspective for the event has been cached, obtaining the global perspective for the event based on the cached global perspective; and

when the global perspective for the event has not been cached, obtaining the global perspective for the event from a community database.

6. The method of claim **5**, further comprising:

determining whether the cached global perspective for the event has expired, based on a time-to-live value associated with the cached global perspective; and

when the global perspective for the event has expired, obtaining the global perspective for the event from the community database.

7. The method of claim **4**, further comprising:

heuristically displaying the information identifying the first object performing the action, the information identifying the action being performed, the information identifying the target upon which the act is being performed, and information associated with the global perspective according to one or more of:

a location associated with the first object;

at least one characteristic of an outbound connection associated with the first object;

a dropped executable file;

an age of the first object; and

a number of concurrent connections associated with the first object.

8. The method of claim **1**, wherein the global perspective comprises information comprises at least one of:

an age;

a popularity;

a determination as to whether the first object is malware;

a determination as to whether the originating object is malware;

an Internet Protocol (IP) Address; and

a Uniform Resource Locator (URL).

9. The system of claim **8**, wherein the sensor agent is further operable to:

cache the event and one or more other events; and

asynchronously consolidate and normalize the event and the one or more other events.

10. A system for providing forensic visibility into a system, comprising:

a device comprising:

a communication interface;

a processor;

data storage; and

a sensor agent stored on the data storage that is executable by the processor, wherein the sensor agent is operable to:

gather an event defining an action of a first object acting on a target;

generate contextual state information for the event by correlating the event to an originating object of the first object;

obtain a global perspective for the event based on the contextual state information, wherein the global perspective comprises information associated with one or more of the first object and the originating object, and wherein the global perspective relates to one or more other events related to the event across a network;

generate an event line comprising information relating to the event, wherein the information relates to at least one of the first object, the action of the first object, the target, and the originating object; and

transmit the generated event line utilizing the communication interface.

11. The system of claim 10, wherein the event is executed by the processor.

12. The system of claim 10, further comprising:

a global perspective information server; and

an end recipient device, wherein the information associated with the one or more of the first object and the originating object is obtained from the global perspective information server and the sensor agent transmits the event line to the end recipient device.

13. The system of claim 10, wherein the event includes information identifying the first object performing the action, information identifying the action being performed, and information identifying the target upon which the act is being performed.

14. The system of claim 13, wherein the sensor agent is further operable to:

determine whether the global perspective for the event has been cached;

when the global perspective for the event has been cached, obtain the global perspective for the event based on the cached global perspective; and

when the global perspective for the event has not been cached, obtain the global perspective for the event from a community database.

15. The system of claim 14, wherein the sensor agent is further operable to:

determine whether the cached global perspective for the event has expired, based on a time-to-live value associated with the cached global perspective; and

when the global perspective for the event has expired, obtain the global perspective for the event from the community database.

16. The system of claim 13, wherein the end recipient device heuristically displays the information identifying the first object performing the action, the information identifying the action being performed, the information identifying the target upon which the act is being performed, and information associated with the global perspective according to one or more of:

a location associated with the first object;

at least one characteristic of an outbound connection associated with the first object;

a dropped executable file;

an age of the first object; and

a number of concurrent connections associated with the first object.

17. A system for providing forensic visibility into a system, comprising:

a first device comprising:

a communication interface;

a processor;

data storage; and

a sensor agent stored on the data storage that is executable by the processor, wherein the sensor agent is operable to:

gather an event defining an action of a first object acting on a target;

determine a global perspective for the event based on contextual state information for the event, wherein the global perspective relates to one or more other events related to the event across a network, and wherein the contextual state comprises an indication of an originating object of the first object and an indication of at least one of a second device on which the first object is executed and a user associated with the first object;

generate an event line comprising information relating to the event, wherein the information relates to at least one of the first object, the action of the first object, the target, and the originating object; and

transmit the generated event line utilizing the communication interface.

18. The system of claim 17, further comprising a global perspective information server comprising:

a processor;

a communication interface;

data storage; and

a global perspective module, wherein the global perspective module is operable to:

receive the event line from the device;

assemble a second event line including details associated with the event, wherein the second event line comprises information identifying the first object, information identifying the action of the first object, information identifying the target, and information identifying the originating object; and

transmit the second event line utilizing the communication interface to an end recipient.

19. The system of claim 18, wherein the end recipient heuristically displays the information identifying the first object, the information identifying the action of the first object, the information identifying the target, and information associated with the global perspective according to one or more of:

a location associated with the first object;

at least one characteristic of an outbound connection associated with the first object;

a dropped executable file;

an age of the first object; and

a number of concurrent connections associated with the first object.

20. The system of claim 17, further comprising a bridge comprising:

a processor;

a communication interface; and

a cached event store, wherein the cached event store is operable to:

receive a first request for a global perspective for the event;

receive a second request for a global perspective for another event, the second request originating from another device;

obtain a first global perspective for the event by obtaining information associated with one or more of the first object and the originating object;

obtain a second global perspective for the another event;

provide the first global perspective to the device; and

provide the second global perspective to the another device.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 10,257,224 B2
APPLICATION NO. : 15/437275
DATED : April 9, 2019
INVENTOR(S) : Joseph A. Jaroch et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

On the Title Page

(72) Inventors should read: Joseph A. Jaroch, Deer Park, IL (US);
Jacques Etienne Erasmus, Belper (GB);
Paul Barnes, Derby (GB);
Johannes Mayr, Linz (AT);
Michael Leidesdorff, Niwot, CO (US);
Marco Giuliani, Bastia Umbra (IT);
Christopher Jon Williams, Derbyshire (GB);
Chad Edward Bacher, Arvada, CO (US)

In the Specification

Column 1, Line 9, "patent application Ser. No. 14/270,069, filed May 5, 2014," should be
-- patent application Ser. No. 14/270,069 (now U.S. Patent No. 9,578,045), filed May 5, 2014, --

Signed and Sealed this
Twenty-first Day of May, 2019

Andrei Iancu

Andrei Iancu
Director of the United States Patent and Trademark Office

Exhibit 5

U 8164950

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

October 07, 2021

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THIS OFFICE OF:

U.S. PATENT: *10,284,591*

ISSUE DATE: *May 07, 2019*

By Authority of the

Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office

R GLOVER

Certifying Officer

US010284591B2

(12) **United States Patent**
Giuliani et al.

(10) **Patent No.: US 10,284,591 B2**
(45) **Date of Patent: May 7, 2019**

(54) **DETECTING AND PREVENTING EXECUTION OF SOFTWARE EXPLOITS**

(71) Applicant: **Webroot Inc.**, Broomfield, CO (US)

(72) Inventors: **Marco Giuliani**, Bastia Umbra (IT); **Marco Bizzarri**, Sassoferrato (IT); **Benedetto Voltattorni**, San Benedetto del Tronto (IT); **Johannes Mayr**, Linz (AT)

(73) Assignee: **WEBROOT INC.**, Broomfield, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 160 days.

(21) Appl. No.: **14/606,604**

(22) Filed: **Jan. 27, 2015**

(65) **Prior Publication Data**

US 2015/0215335 A1 Jul. 30, 2015

Related U.S. Application Data

(60) Provisional application No. 61/931,772, filed on Jan. 27, 2014.

(51) **Int. Cl.**

H04L 29/06	(2006.01)
G06F 21/55	(2013.01)
G06F 21/52	(2013.01)

(52) **U.S. Cl.**
CPC ***H04L 63/1441*** (2013.01); ***G06F 21/52*** (2013.01); ***G06F 21/554*** (2013.01); ***H04L 63/1408*** (2013.01)

(58) **Field of Classification Search**
CPC . H04L 63/1408; H04L 63/1441; G06F 21/52; G06F 21/554
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,313,821 B1 * 12/2007 Steiner G06F 21/554 714/E11.207
8,037,526 B1 * 10/2011 Satish G06F 21/52 711/6

(Continued)

OTHER PUBLICATIONS

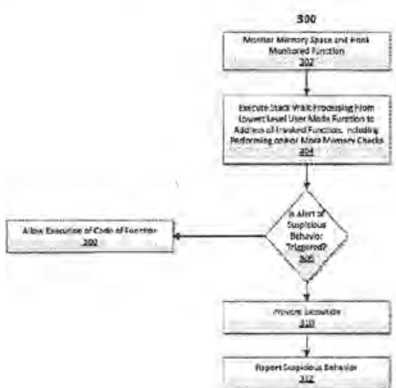
International Search Report and Written Opinion for International Application No. PCT/US15/13111, dated Apr. 17, 2015, 15 pages.

Primary Examiner — Joseph P Hirl
Assistant Examiner — Thomas A Gyorfi
(74) *Attorney, Agent, or Firm* — Merchant & Gould P.C.

(57) **ABSTRACT**

In non-limiting examples, anti-exploit systems and methods described herein monitor a memory space of a process for execution of functions. Stack walk processing is executed upon invocation of one of the functions in the monitored memory space. During execution of the stack walk processing, at least one memory check is performed to detect suspicious behavior. An alert of suspicious behavior is triggered when the performing of the memory check detects at least one of: code execution attempted from non-executable memory, identification of an invalid base pointer, identification of an invalid stack return address, attempted execution of a return-oriented programming technique, the base pointer is outside a current thread stack, and a return address is detected as being inside a virtual memory area. If an alert of suspicious behavior is triggered, execution of a payload is prevented for the invoked function.

18 Claims, 4 Drawing Sheets



Copy provided by USPTO from the PIRS Image Database on 10-05-2021

(56) **References Cited**

U.S. PATENT DOCUMENTS

8,510,596	B1	8/2013	Gupta et al.	
2005/0108562	A1	5/2005	Khazan et al.	
2006/0230388	A1 *	10/2006	Hatlelid	G06F 21/52 717/127
2007/0016914	A1 *	1/2007	Yeap	G06F 21/554 719/328
2007/0089088	A1 *	4/2007	Borde	G06F 21/52 717/106
2009/0049550	A1 *	2/2009	Shevchenko	G06F 21/56 726/23
2011/0145924	A1	6/2011	Kolsek et al.	
2011/0289586	A1 *	11/2011	Kc	G06F 21/566 726/24
2012/0255013	A1	10/2012	Sallam	
2012/0255018	A1	10/2012	Sallam	
2013/0198527	A1 *	8/2013	Lu	G06F 9/445 713/189

* cited by examiner

FIG. 1

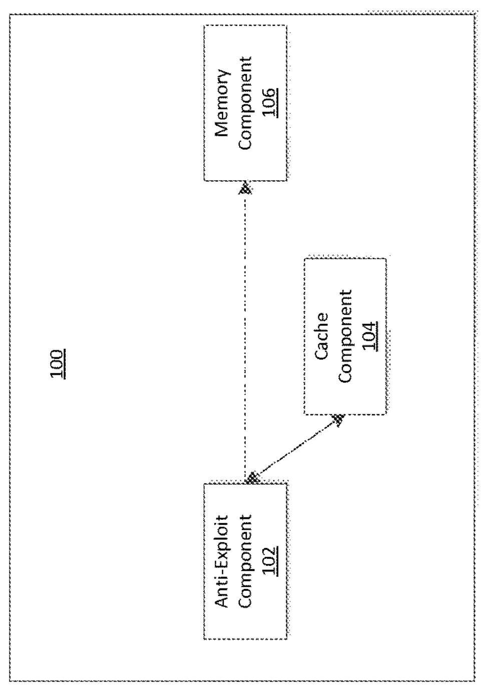

100
Anti-Exploit Component 102
Cache Component 104
Memory Component 106

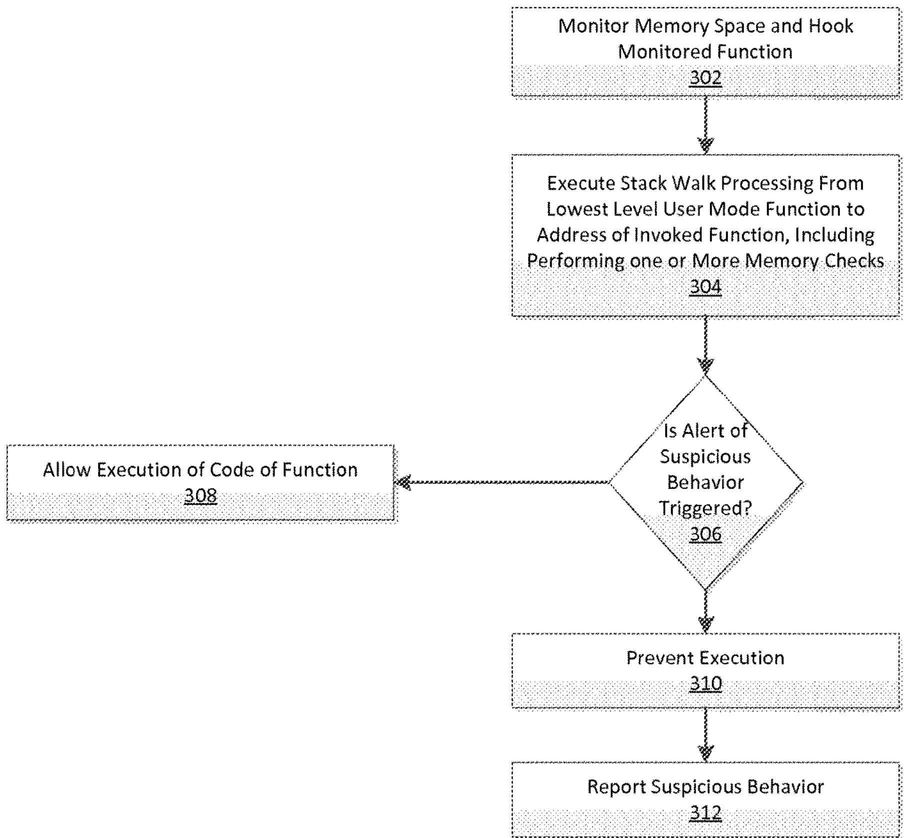
FIG. 2

200

Timing Associated with Execution of Memory Check 202	Memory Check Executed During Stack Walk Processing For Called Function 204
Before Address is Reached for Originating Caller Function 206	• Is Code Execution Attempted From Non-Executable Memory? • Is Base Pointer Identified As Being Invalid? • Is An Invalid Stack Return Address Identified? • Is Execution Of a Return-Oriented Programming Technique Attempted? • Is Base pointer Detected As Being Outside a Current Thread Stack? • Is Return Address Detected as Being Inside a Virtual Memory Area? Block 210
Once Address is Reached for Originating Caller Function 208	• Is Execution Attempted for Originating Caller Function/ Low Level User Mode Function in Cache? Block 212

FIG. 3

300



WBRT007981

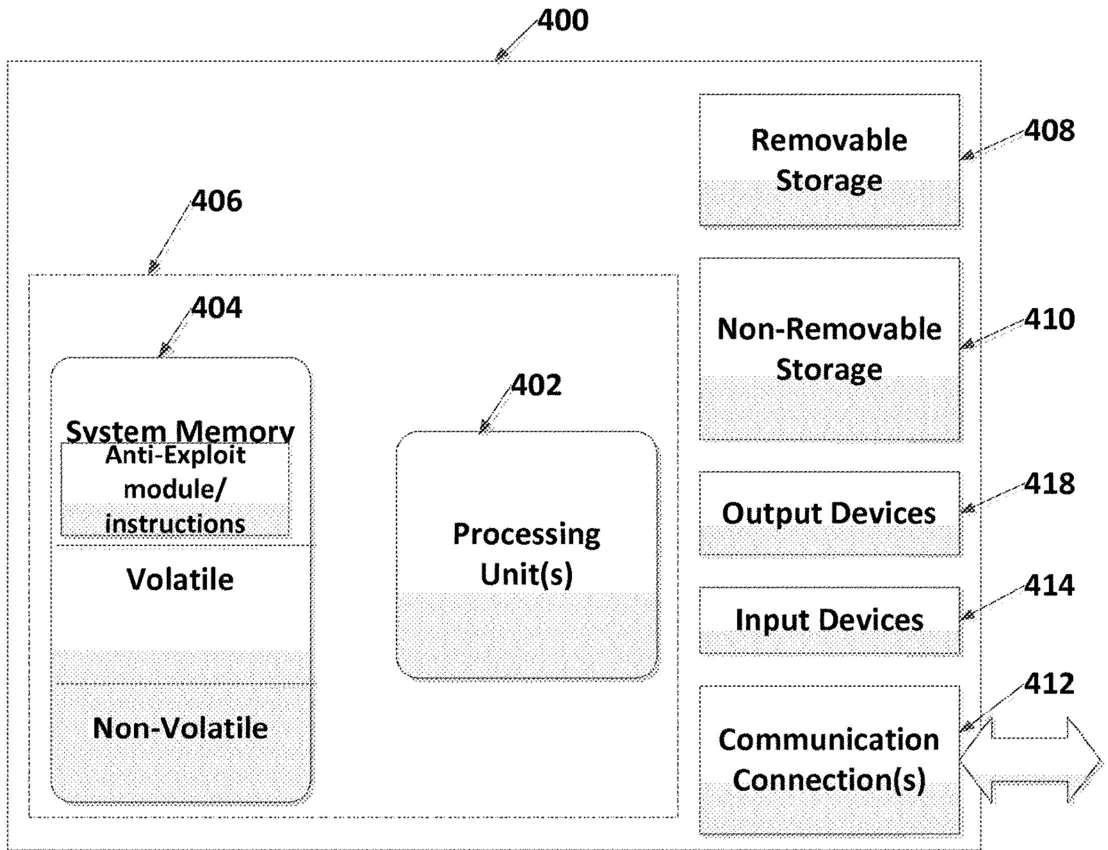

400
Removable Storage
408
406
Non-Removable Storage
410
404
402
System Memory
Anti-Exploit module/ instructions
Output Devices
418
Processing Unit(s)
Volatile
Input Devices
414
412
Communication Connection(s)
Non-Volatile

FIG. 4

DETECTING AND PREVENTING EXECUTION OF SOFTWARE EXPLOITS

PRIORITY

This application claims the benefit of U.S. Provisional Application No. 61/931,772, filed on Jan. 27, 2014, which is hereby incorporated by reference in its entirety.

BACKGROUND

An exploit is a piece of code, software, data, or a sequence of commands that takes advantage of a bug, glitch or vulnerability in order to cause unintended or unanticipated behavior to occur on computer software, hardware, or any other electronic component. Exploits are usually intended for malicious purposes and pose threats that can compromise a computing system or device, for example, by hijacking normal code flow execution of a process towards a zone of memory where exploit code currently stands. For instance, exploit code may launch a new process or manipulate an existing process to perform malicious execution. Prevention of software exploits from executing code is a paramount security concern for system users and administrators. It is with respect to this general environment that aspects of the present technology disclosed herein have been contemplated.

SUMMARY

Aspects of the present disclosure relate to anti-exploit examples. In non-limiting examples, anti-exploit systems and methods described herein monitor a memory space of a process for execution of functions. Stack walk processing is executed upon invocation of one of the functions in the monitored memory space. During execution of the stack walk processing, at least one memory check is performed to detect suspicious behavior. An alert of suspicious behavior is triggered when the performing of the memory check detects at least one of: code execution attempted from non-executable memory, identification of an invalid base pointer, identification of an invalid stack return address, attempted execution of a return-oriented programming technique, the base pointer is outside a current thread stack, and a return address is detected as being inside a virtual memory area. If an alert of suspicious behavior is triggered, execution of a payload is prevented for the invoked function.

In other non-limiting examples, the anti-exploit examples described herein monitor a memory space of a process for execution of functions. Stack walk processing is executed upon invocation of one of the functions in the monitored memory space. During execution of the stack walk processing, at least one memory check is performed to detect suspicious behavior. An alert of suspicious behavior is triggered when the performing of the memory check detects at least one of: code execution attempted from non-executable memory, identification of an invalid base pointer, identification of an invalid stack return address, attempted execution of a return-oriented programming technique, the base pointer is outside a current thread stack, a return address is detected as being inside a virtual memory area, and a low level user mode function is attempted to be executed in a cache, where the cache is evaluated for identification of an address of the invoked function. One of skill in the art will appreciate that the embodiments disclosed herein can be employed to detect other types of suspicious behavior. If an alert of suspicious behavior is triggered, action is taken to prevent execution of a payload for the invoked function.

This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used to limit the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

Non-limiting and non-exhaustive examples are described with reference to the following figures. As a note, the same number represents the same element or same type of element in all drawings.

FIG. **1** illustrates an exemplary system **100** showing interaction of components for implementation of exemplary anti-exploit examples described herein.

FIG. **2** illustrates an exemplary chart **200** of one or more memory checks performed by the anti-exploit examples described herein.

FIG. **3** illustrates an exemplary method **300** for implementation of anti-exploit systems and methods described herein.

FIG. **4** illustrates one example of a suitable operating environment **400** in which one or more of the present examples may be implemented.

DETAILED DESCRIPTION

Non-limiting examples of the present disclosure provide anti-exploit examples that prevent exploit code from being executed. Examples describe monitoring of memory process control flow in order to detect potential suspicious behavior and performing at least one check to evaluate how memory is being allocated before setting/marking an area of memory space as suspicious. By analyzing characteristics of memory space where a function is called from, the anti-exploit examples are able to effectively and accurately determine whether an exploit is attempting to execute its own code or if intended execution of application code is normal system behavior.

A number of technical advantages are achieved based on the present disclosure including but not limited to: enhanced security protection, improved detection of potential security exploits, reduction in error rate in identification and marking of suspicious behavior (e.g., false positives), and improved usability and interaction with users who are not required to continuously monitor for security exploits, among other examples.

FIG. **1** illustrates an exemplary system **100** showing interaction of components for implementation of exemplary anti-exploit technology described herein. Exemplary system **100** presented is a combination of interdependent components that interact to form an integrated whole for execution of exploit detection and prevention operations. Components of the systems may be hardware components or software implemented on and/or executed by hardware components of the systems. In examples, system **100** may include any of hardware components (e.g., used to execute/run operating system (OS)), software components (e.g., applications, application programming interfaces, modules, virtual machines, runtime libraries, etc.) running on hardware. In one example, an exemplary system **100** may provide an environment for software components to run, obey constraints set for operating, and makes use of resources or

facilities of the system **100**, where components may be software (e.g., application, program, module, etc.) running on one or more processing devices. For instance, an anti-exploit mechanism (e.g., application, instructions, modules, etc.) may be run on a processing device such as a computer, mobile device (e.g., smartphone/phone, tablet) and any other electronic devices, where components of the system may be executed on the processing device. In other examples, the components of systems disclosed herein may be spread across multiple devices. For instance, input may be entered on a client device (e.g., mobile phone) and information may be processed or accessed from other devices in a network such as one or more server devices used to implement an anti-exploit mechanism or protocol.

As one example, the system **100** comprises an anti-exploit component **102**, a cache component **104**, and a memory component **106**, each having one or more additional components. The scale of systems such as system **100** may vary and include more or less components than those described in FIG. **1**. In some examples, interfacing between components of the system **100** may occur remotely, for example where anti-exploit technology is implemented remotely to monitor and control process flow for detection of suspicious behavior (e.g., exploit) and prevention of execution of code of the suspicious behavior.

An anti-exploit component **102** implements anti-exploit examples described herein to detect suspicious behavior and prevent exploit code of software exploits from being executed. An exploit is a piece of code, software, data, or a sequence of commands that takes advantage of a bug, glitch or vulnerability in order to cause unintended or unanticipated behavior. Example types of exploits include malware such as computer viruses, worms, trojan horses, spyware and adware, among other examples. As an example, the anti-exploit examples described herein prevent exploit code of a process from executing another process or loading a module into the process memory. The anti-exploit examples implemented by the anti-exploit component **102** are able to be configured to detect and prevent software exploits that attempt to execute another process or load a module into process memory, for instance, a process where shellcode is able to be injected and executed.

An action of calling a system process (e.g., function of an application or application programming interface (API)) from a low level perspective, involves the presence of a particular set of assembly instruction, such as a CALL, inside the process memory. A low level operation or instruction provides minimal abstraction from a computer's instruction set architecture, for example either machine code or assembly language. The word "low" refers to the small or nonexistent amount of abstraction between the programming language and machine language. Low-level languages can be converted to machine code without using a compiler or interpreter, and the resulting code runs directly on the processor. Once the CALL instruction is reached, the code flow of the program starts to follow a chain of functions which will lead to the lowest level user mode function, before transitioning to Kernel Mode. In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. In user mode, the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory. Most of the code running on a processing device such as a computer will execute in user mode. Thus, an instruction from user mode involves system call instructions (e.g., triggering operating system dependent software interrupt or control transfer instructions such as a sysenter assembly instruction) to execute a kernel function.

Generally, a non-malicious call starts from a range of pages in a zone of memory containing application or system module execution code. This zone, of memory contains data (e.g., pages) used to store a module (e.g., executable file (.exe) or dynamic-link library (DLL)) assembly code. The zone of memory may share physical storage with other processes. As an example, the zone of memory may be marked as MEM_IMAGE. The anti-exploit described herein detect functions called from user mode, for example, that attempt to allocate new pages of memory, reallocate system memory or modify the type of a memory page. In examples, an exploit may be initiated from another zone of memory (e.g., outside of the MEM_IMAGE zone/region such as a non MEM_IMAGE region). In another example, an exploit may attempt alter instruction sequences within a system memory region such as MEM_IMAGE, for example, by return-oriented programming (ROP). The anti-exploit component **102** monitors for, detects, and takes action to prevent such suspicious behavior.

In order to reach a goal of starting a new process, an exploit has to execute a complete execution stack starting from the shellcode and running down through to a lowest level user function (e.g., instruction that includes machine or assembly instruction). Shellcode is a small piece of code used as the payload in the exploitation of a software vulnerability. An exploit will commonly inject shellcode into the target process before or at the same time as it exploits a vulnerability to gain control over a program counter. The program counter is adjusted to point to the shellcode, after which the shellcode (including the malicious payload) is executed.

The anti-exploit component **102** executes a function hooking technique to hook to the lowest level user mode function and intercept system routines of a called API. A hooking technique covers a range of techniques used to alter or augment the behavior of an operating system, of applications, or of other software components by intercepting function calls or messages or events passed between software components. A hook is code that handles such intercepted function calls, events or messages.

The anti-exploit component **102** loads a component such as a custom module into the memory space of each monitored process. This component intercepts monitored operating system applications or APIs and evaluates a record (e.g., stack frame) of a call stack. A stack frame is a frame of data that gets pushed onto a stack that stores information about the active subroutines of a computer program. As an example, a stack frame may represent a function call and its argument data.

When unidentified code tries to allocate/mark a new executable page of memory which is not related to any valid process module, the custom component monitors these newly allocated memory regions and prevents any attempted execution of monitored functions. Functions monitored comprise processes that attempt execution of another process or loading of a module into a process memory, for example, function such as NtCreateProcess, NtCreateProcessEX, NtCreateUserProcess, LdrLoadDll, among other examples monitored functions. The present application is not limited to functions of a specific operating system. One skilled in the art will appreciate that the embodiments disclosed herein can be employed to cover functions that attempt execution of another process or loading of a module

into a process memory, implementable by/in other operating systems. The anti-exploit examples of the anti-exploit component **102** are configurable to be programmed to monitor any type of function of an application/API. A list of monitored functions may be maintained by the anti-exploit component **102**. Examples of functions monitored by the anti-exploit technology include but are not limited to: NtAllocateVirtualMemory, NtProtectVirtualMemory, NtCreateProcess, NtCreateProcessEX, NtCreateUserProcess, LdrLoadDll, among other examples. Again, one skilled in the art will appreciate that the embodiments disclosed herein can be employed to monitor any functions that attempt execution of another process or loading of a module into a process memory, implementable by/in other operating systems. The list of monitored functions may change perpetually and can be updated (e.g., by an administrator or software update) at any time to improve the robustness of the functions monitored and improve security against software exploits. The function hooking technique of the anti-exploit component **102** intercepts the system routines of the called process.

A hooked function (including but not limited to NtCreateProcess, NtCreateProcessEX, NtCreateUserProcess, LdrLoadDll) is monitored/evaluated through execution of stack walk processing. The anti-exploit component **102** executes stack walk processing that walks up a call stack for the process (function) until an address of the first function of a called function is reached. As an example, for stack walk processing, when one of the functions listed before is invoked, the first step is to get the value of the call frame pointer for being able to walk the call stack frame. The call frame pointer contains the address of the stack frame for the called function. In the examples disclosed herein, the anti-exploit technology may go back through the stack_entry chain until the initial caller's stack frame is reached (e.g., back-walk the stack frame starting from the lowest level user mode function to an address where the call sequence started for the hooked function). Once reached, an address from where the call sequence has started for the hooked function is located and characteristics of that particular memory region are retrieved and analyzed (e.g., evaluating state information of the particular memory region). In the stack walk processing, an address of a base pointer of the stack and an address of a stack pointer are identified for a stack frame associated with the called function. As an example of a base pointer, a precFp may be identified. A precFp is a pointer to the previous STACK_ENTRY structure of the called function. The retAddr is the address where the control flow must continue after the called function has completed. Each time a CALL is executed, a new stack frame is created and the return address is set with the value of the instruction following CALL opcode (e.g., portion of machine language instruction that specifies an operation to be performed).

During the stack walk processing of the stack frame, a first call address of a call sequence is one that is present in the last valid stack entry. A list of conditions may be evaluated against each stack entry found during the stack walk. As examples, a stack frame of a called function may be required to satisfy conditions such as:

- both, precFp and retAddr, must be valid accessible addresses (verified via system APIs, for example VirtualQuery( ) and TestVirtualMemoryAccess( )); and
- if these addresses are valid, then the previous 10 bytes before the retAddr address are parsed, searching for a CALL assembly instruction of the hooked function. Every time a CALL instruction is executed, a new stack frame is created and the return address is set with the value of the address of the instruction following the CALL op-code.

In a case where conditions, such as the above identified conditions, are satisfied, the stack walk processing may proceed with evaluating a memory region associated with a called function. For example, stack walk processing may include performing one or more memory checks from the lowest level user function of the hooked function down through the address of the originating caller function (e.g., the hooked function). Once the address is located from where the call sequence has started, characteristics of that particular memory region are retrieved and analyzed. In analyzing the characteristics of the memory region, the memory region data (e.g., memory pages) maintained in a cache by the anti-exploit examples disclosed herein are evaluated during performance of a memory check. The stack walk processing executed by the anti-exploit component **102** performs a memory check or checks on a frame of call stack pertaining to the called process (including one or more functions) attempting to detect and identify suspicious behavior that may pertain to a software exploit. Examples of memory checks performed are provided in a later description of FIG. **2**. Unlike other exploit prevention technologies that try to prevent that an exploit starts its own shellcode, the anti-exploit examples implemented by the anti-exploit component **102** prevent that the shellcode performs an ultimate execution of a malicious payload, whether the malicious payload is loading another module or executing another process.

The anti-exploit component **102** interfaces with a cache component **104** as shown in system **100** of FIG. **1**. The cache component **104** is a structure (hardware or software) that stores data related to a memory region for a monitored process. Memory used for a cache is allocated by the anti-exploit technology, for example by the anti-exploit component **102**. The cache component **104** maintains a cache for a monitored process. A cache is a specific memory region allocated in each monitored process where information about monitored memory pages is tracked. In an example, the cache component **104** interface with the anti-exploit component **102** during hooking of a monitored function to store state information about pages of memory associated with the called function when the permission is changed, for example, changing the permission to "execute." State information tracked by the cache component **104** comprises information about the allocation of memory pages for a monitored process and the initial permissions and type of the memory pages as well as any later changes of permissions for each memory page. In at least one variation of system **100**, the cache component **104** may be included in the anti-exploit component **102**. While FIG. **1** illustrates the cache component **104** as a separate component from the anti-exploit component **102** for ease of explanation, the anti-exploit technology implemented may be a collection of operations that include the ability to cache data.

During execution of the stack walk processing, the cache component **104** tracks and evaluates the state information about pages of memory relate to a monitored process. The hooked function is triggered and a back-walk of the stack frame is initiated, where the back-walk walks up the call stack from the lowest level user mode function (hooked function) until the first function of the chain is reached. The anti-exploit component **102** interfaces with the cache component **104** to evaluate the tracked memory information for a monitored process, for example while a memory check is

being performed during the stack walk processing. The tracked memory information in the cache component **104** is evaluated before the anti-exploit technology allows code of the process to access memory space (e.g., system memory) of the memory component **106**. Memory checks performed pertaining to tracked data in the cache may trigger an alert of suspicious behavior, for example, where the anti-exploit component **102** identifies what it believes to be shellcode and prevents execution of a malicious payload of the shellcode.

The cache component **104** is implementable as a linked list of nodes. Each node entry stores specific data (including the nature of the potential attack) about a memory region that is monitored. A new node may be added any time system service routines are invoked related to a memory space. Examples of system services routines associated with memory include functions similar to NtAllocateVirtualMemory or NtProtectVirtualMemory.

As an example, a function that may be monitored is a function associated with creation or allocation of at least one region in a memory space (e.g., virtual memory space). For instance, NtAllocateVirtualMemory is a routine invoked in order to reserve, commit, or both, a region of pages within the user-mode virtual address space of the process. As an example, a new node is created if the memory is allocated as PAGE_EXECUTE_READWRITE in a non MEM_IMAGE region or in case a suspicious behavior matching logic rules was found during the stack walk.

As another example, a function associated with protection or change to protection in at least one region of a memory space (e.g., virtual memory space) may be monitored. For instance, NtProtectVirtualMemory is a routine invoked in order to change the protection on a region of committed pages of the virtual address space of the process. As an example, a new node is created if the memory protection of a non MEM_IMAGE region is changed to PAGE_EXECUTE_READWRITE and the prior protection/permission assigned to the page was non-executable or in case a suspicious behavior matching logic rules was found during the stack walk.

When a memory check performed during the stack walk processing does not trigger an alert of suspicious behavior, the anti-exploit component **102** enables a complete called function to access the a system kernel via the memory component **106**. The memory component **106** is one or more structures (hardware or software) of system memory. In a case where a memory check performed during the stack walk processing triggers an alert of suspicious behavior (e.g., a potential security exploit), the anti-exploit component **102** prevents the execution of the payload of the called function, thus preventing access to the memory component **106**.

FIG. **2** illustrates an exemplary chart **200** of memory checks performed by the anti-exploit examples described herein. Chart **200** identifies exemplary memory checks performed during stack walk processing. As referenced above, a stack-walk is executed in reverse order from a system call-up to an initiating caller address, which could be shellcode. By performing the memory checks during the stack walk processing (e.g., back-walk of the stack frame), shellcode is able to be prevented from performing execution of a malicious payload.

In chart **200**, column **202** identifies a timing associated with performance of exemplary memory checks and comprises rows **206** and **208**. Row **206** identifies a timing of performing memory checks before an address is reached for an originating caller function. Memory checks performed during the stack walk processing at a timing before an address is reached for an originating caller function (row **206**), include the memory checks identified in block **210**. Row **208** identifies a timing of performing memory checks once an address is reached for an originating caller function. Memory checks performed during the stack walk processing once an address is reached for an originating caller function (row **208**), include the memory checks identified in block **212**.

Memory checks (identified in block **210**) are made continuously during the stack walk processing until the address of an originating caller function (e.g., hooked function) is reached. At least one additional memory check (block **212**) is made if the address of the originating caller function is reached. An example of an originating caller function is the hooked and monitored function that is associated with the stack frame being evaluated.

Memory checks (column **204**) performed at a timing (column **202**) before an address for an originating caller function is reached (as indicated in row **206**) are shown in block **210**. Block **210** includes, but is not limited to, the following memory checks:

- Is code execution attempted from non-executable memory;
- Is a base pointer of a stack frame identified as being invalid;
- Is invalid stack return address identified for a stack frame;
- Is attempted execution of a return-oriented programming technique detected;
- Is the base pointer of a stack frame detected as being outside a current thread stack, and
- Is a return address detected as being inside a virtual memory area.

One of skill in the art will appreciate that a subset of these checks or other checks not listed may be performed without departing from the spirit of this disclosure.

For detection of code execution from non-executable memory as shown in block **210**, the systems and methods disclosed herein determine whether a process is being loaded from a memory address of the memory space that is marked as non-executable or if a return address found in the stack walk chain falls in a non-executable memory. As an example, the anti-exploit examples may look for flags indicating that a process is associated with non-executable memory of the memory space. For instance, flags indicating functions similar to PAGE_EXECUTE, PAGE_EXECUTE_READ, PAGE_EXECUTE_READWRITE or PAGE_EXECUTE_WRITECOPY, can trigger an alert of suspicious behavior.

For detection of an invalid base pointer (e.g., base stack pointer) as shown in block **210**, anti-exploit systems and methods evaluate whether a memory address associated with a base pointer of a stack frame fails into a memory address that is not accessible or not allocated. For example, if a base pointer for a current stack frame (e.g., current EBP) during the stack walk falls into a memory address that is not accessible or not allocated, an alert of suspicious behavior is triggered. In a case where an invalid base pointer is found inside a virtual memory management API, the address will be saved in the local cache and further evaluated using an additional memory check such as whether execution of an originating caller function is attempted in the cache once the address for the originating caller function is reached in the stack walk processing. If instead the behavior is found from an API process such as or similar to loadImage/createprocess, an alert of suspicious behavior will be triggered immediately.

An alert of suspicious behavior is triggered when an invalid stack return address is detected as shown in block **210**. Anti-exploit systems and methods evaluate if the current stack pointer during the stack walk falls into a memory address that is not accessible. If so, an alert is triggered. In the case an invalid stack return address is found inside a virtual memory management API, the address will be saved in the local cache. If instead the behavior is found from an API process such as or similar to loadImage/createprocess, an alert of suspicious behavior will be triggered immediately.

If the current return address during the stack walk is a ROP gadget as shown in block **210**, anti-exploit systems and methods trigger an alert of suspicious behavior. A ROP gadget is a security exploit that executes carefully chosen machine instruction sequences. Some exploit scenarios describe previously relate to injecting code into an API and getting the injected code executed. In evaluating a ROP gadget, this memory check evaluates whether API code (without injecting code) is to be executed out of sequence, for example, where a ROP gadget may force API code to jump to another place of code in an order intended to exploit the API code. In a case where a ROP gadget is found inside a virtual memory management API, the address will be saved in the local cache and further evaluated using an additional memory check such as whether execution of an originating caller function is attempted in the cache once the address for the originating caller function is reached in the stack walk processing. If instead the behavior is found from an API process such as or similar to loadImage/createprocess, an alert of suspicious behavior will be triggered immediately.

In some cases, a base pointer may be valid and accessible outside a current thread stack for a stack frame being evaluated as shown in block **210**. In examples, when a base pointer is outside a current thread stack an alert for suspicious behavior is triggered. In the case where a base pointer is outside a current thread stack but inside a virtual memory management API, the address will be saved in the local cache and further evaluated using an additional memory check such as determining whether execution of an originating caller function is attempted in the cache once the address for the originating caller function is reached in the stack walk processing. If instead the behavior is found from an API process such as or similar to loadImage/createprocess, an alert of suspicious behavior will be triggered immediately.

Another memory check shown in block **210** is performed to evaluate whether a return address is detected as being inside a virtual memory area that is going to be marked or allocated as executable. If the current return address during the stack walk chain falls into the memory area which is going to be allocated/marked as executable an alert will be triggered immediately.

Once the address of the originating caller function is reached in the stack walk processing (as identified in row **208**), an additional check is performed to evaluate the address of the invoked/called function found in a cache entry is executable as shown in block **212**. During the stack walk processing, an address of an invoked function (e.g., originating caller function is identified). If an address of an invoked function is found in a cache entry, then the call is suspicious and an alert is triggered. Each entry present in the cache structure represents a part of memory that has been recently allocated with as executable (e.g., by a flag such as or similar to "EXECUTE"), and is thus a potential piece of memory that could host malicious code. If the anti-exploit technology intercepts a call to API functions such as or similar to NtCreateProcess/NtCreateProcessEx/LdrLoadDll from one of such entries, it automatically marks this behavior as suspicious and potentially malicious.

FIG. **3** illustrates an exemplary method **300** for implementation of anti-exploit systems and methods described herein. As an example, method **300** may be executed by an exemplary system such as system **100** of FIG. **1**. In other examples, method **300** may be executed on a device comprising at least one processor configured to store and execute operations, programs or instructions. However, method **300** is not limited to such examples. In other examples, method **300** may be performed an application or service executing the anti-exploit technology.

Method **300** begins at operation **302** where a memory space of a process is monitored for execution of at least one of a plurality of functions. Examples of functions monitored are described as part of the description of FIG. **1**. As an example, operation **302** monitors functions associated with memory management in the memory space by a user mode function (e.g., a call initiated from a user mode). In at least one example, operation **302** further comprises applying, upon detecting a function call of one of the plurality of functions in the monitored memory space, a function hooking technique to hook a lowest level user mode function of a called process (e.g., API). In examples, the monitoring may further comprise detecting an attempt by unidentifiable code to allocate a new executable page of memory and monitoring the allocated new executable page of memory.

Flow proceeds to operation **304** where stack walk processing is performed on a hooked and/or monitored function. When one of the plurality of functions is invoked, the stack walk processing is executed for a stack frame associated with an invoked function (e.g., originating caller function). Execution of the stack walk processing is described in further detail in the description of FIG. **1** and is a process that walks a stack frame for a hooked functioned from a lowest level user mode function to an address of the invoked function/originating caller function. Operation **304** further comprises performing one or more memory checks during the stack walk processing. Examples of memory checks that are performed are described in detail in the description of FIG. **2**. In one example, an originating address associated with the invoked function is identified.

Flow proceeds to decision operation **306** where a determination is made as to whether an alert of suspicious behavior is triggered based on execution of the one or more memory checks performed during the stack walk processing. For example, chart **200** of FIG. **2** identifies timing associated with performance of specific memory checks. In some examples, an alert of suspicious behavior is triggered before an address of an originating caller function is reached in the stack walk processing. In other examples, each memory check is performed before an alert is triggered. If decision operation **306** determines that no alert of suspicious behavior is triggered, then flow proceeds to operation **308** where execution of code (e.g., a payload) associated with an invoked function is allowed. If decision operation **306** determines that an alert of suspicious behavior is triggered, the flow proceeds to operation **310** where execution of code associated with an invoked function is prevented. As an example, operation **310** takes action to prevent execution of a payload for the invoked function. In some example, method **300** comprises an operation of reporting suspicious behavior (operation **312**). Data collected from tracking identification of potentially malicious behavior and detection of malicious behavior (as well as successful cases where

execution of malicious behavior was prevented) may be useful in tracking security exploits, statistical analysis of the anti-exploit technology, and augmenting anti-exploit technology to improve over time.

FIG. **4** and the additional discussion in the present specification are intended to provide a brief general description of a suitable computing environment in which the present invention and/or portions thereof may be implemented. Although not required, the embodiments described herein may be implemented as computer-executable instructions, such as by program modules, being executed by a computer, such as a client workstation or a server. Generally, program modules include routines, programs, objects, components, data structures and the like that perform particular tasks or implement particular abstract data types. Moreover, it should be appreciated that the invention and/or portions thereof may be practiced with other computer system configurations, including hand-held devices, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

FIG. **4** illustrates one example of a suitable operating environment **400** in which one or more of the present embodiments may be implemented. This is only one example of a suitable operating environment and is not intended to suggest any limitation as to the scope of use or functionality. Other well-known computing systems, environments, and/or configurations that may be suitable for use include, but are not limited to, personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, programmable consumer electronics such as smart phones, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

In its most basic configuration, operating environment **400** typically includes at least one processing unit **402** and memory **404**. Depending on the exact configuration and type of computing device, memory **404** (storing, among other things, anti-exploit module(s), e.g., Anti-exploit applications, APIs, programs etc. and/or other components or instructions to implement or perform the system and methods disclosed herein, etc.) may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.), or some combination of the two. This most basic configuration is illustrated in FIG. **4** by dashed line **506**. Further, environment **400** may also include storage devices (removable, **408**, and/or non-removable, **410**) including, but not limited to, magnetic or optical disks or tape. Similarly, environment **400** may also have input device(s) **414** such as keyboard, mouse, pen, voice input, etc. and/or output device(s) **416** such as a display, speakers, printer, etc. Also included in the environment may be one or more communication connections, **412**, such as LAN, WAN, point to point, etc.

Operating environment **400** typically includes at least some form of computer readable media. Computer readable media can be any available media that can be accessed by processing unit **402** or other devices comprising the operating environment. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media includes volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other non-transitory medium which can be used to store the desired information. Computer storage media does not include communication media.

Communication media embodies computer readable instructions, data structures, program modules, or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of the any of the above should also be included within the scope of computer readable media.

The operating environment **400** may be a single computer operating in a networked environment using logical connections to one or more remote computers. The remote computer may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above as well as others not so mentioned. The logical connections may include any method supported by available communications media. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

The different aspects described herein may be employed using software, hardware, or a combination of software and hardware to implement and perform the systems and methods disclosed herein. Although specific devices have been recited throughout the disclosure as performing specific functions, one of skill in the art will appreciate that these devices are provided for illustrative purposes, and other devices may be employed to perform the functionality disclosed herein without departing from the scope of the disclosure.

As stated above, a number of program modules and data files may be stored in the system memory **404**. While executing on the processing unit **402**, program modules **408** (e.g., applications, Input/Output (I/O) management, and other utilities) may perform processes including, but not limited to, one or more of the stages of the operational methods described herein such as method **300** illustrated in FIG. **3**, for example.

Furthermore, examples of the invention may be practiced in an electrical circuit comprising discrete electronic elements, packaged or integrated electronic chips containing logic gates, a circuit utilizing a microprocessor, or on a single chip containing electronic elements or microprocessors. For example, examples of the invention may be practiced via a system-on-a-chip (SOC) where each or many of the components illustrated in FIG. **4** may be integrated onto a single integrated circuit. Such an SOC device may include one or more processing units, graphics units, communications units, system virtualization units and various application functionality all of which are integrated (or "burned") onto the chip substrate as a single integrated circuit. When operating via an SOC, the functionality described herein

may be operated via application-specific logic integrated with other components of the operating environment **400** on the single integrated circuit (chip). Examples of the present disclosure may also be practiced using other technologies capable of performing logical operations such as, for example, AND, OR, and NOT, including but not limited to mechanical, optical, fluidic, and quantum technologies. In addition, examples of the invention may be practiced within a general purpose computer or in any other circuits or systems.

This disclosure described some aspects of the present technology with reference to the accompanying drawings, in which only some of the possible embodiments were shown. Other aspects may, however, be embodied in many different forms and should not be construed as limited to the embodiments set forth herein. Rather, these aspects were provided so that this disclosure was thorough and complete and fully conveyed the scope of the possible embodiments to those skilled in the art.

Although specific aspects were described herein, the scope of the technology is not limited to those specific embodiments. One skilled in the art will recognize other embodiments or improvements that are within the scope and spirit of the present technology. Therefore, the specific structure, acts, or media are disclosed only as illustrative embodiments. The scope of the technology is defined by the following claims and any equivalents therein.

What is claimed is:

1. A computer-implemented method comprising:

monitoring a memory space of a process for execution of at least one monitored function of a plurality of functions, wherein monitoring the memory space comprises loading a component for evaluating the at least one monitored function in the memory space;

invoking one of the plurality of functions as a result of receiving a call from an application programming instance;

executing stack walk processing upon the invocation of one of the plurality of functions in the monitored memory space; and

performing, during the executing of the stack walk processing before an address of an originating caller function is reached, a memory check for a plurality of stack entries identified during the stack walk processing to detect suspicious behavior, wherein an alert of suspicious behavior is triggered when the performing of the memory check detects at least one of the following:

code execution is attempted from non-executable memory,

a base pointer is identified as being invalid,

an invalid stack return address is identified,

attempted execution of a return-oriented programming technique is detected,

the base pointer is detected as being outside a current thread stack, and

a return address is detected as being inside a virtual memory area,

wherein when an alert of suspicious behavior is triggered, preventing execution of a payload for the invoked function from operating.

2. The method according to claim **1**, further comprising:

identifying an address associated with the invoked function; and

when the address of the invoked function address is found in a cache, preventing execution of a payload for the invoked function.

3. The method according to claim **1**, wherein executing of the stack walk processing further comprises walking a stack frame associated with the invoked function from a low level user mode function to the originating address of the invoked function.

4. The method according to claim **1**, wherein the monitoring further comprises monitoring functions associated with memory management in the memory space by a user mode function.

5. The method according to claim **1**, wherein the monitoring further comprises applying, upon detecting a function call of one of the plurality of functions in the monitored memory space, a hooking technique to hook a lowest level user mode function of the invoked function, wherein the executing of the stack walk processing executes a reverse stack walk from the lowest level user mode function of the invoked function.

6. The method according to claim **1**, wherein the plurality of functions comprises an NtAllocateVirtualMemory function, an NtProtectVirtualMemory function, an NtCreateProcess function, an NtCreateProcessEX function, an NtCreateUserIProcess function, and an LdrLoadDll function.

7. The method according to claim **5**, wherein performing the memory check comprises evaluating state information associated with a cache entry for the hooked function.

8. The method according to claim **7**, wherein when an address of the invoked function is found in a cache, preventing execution of a payload for the invoked function.

9. A system comprising:

at least one memory; and

at least one processor connected with the memory configured to perform operation of:

monitoring a memory space of a process for execution of at least one monitored function of a plurality of functions, wherein monitoring the memory space comprises loading a component for evaluating the at least one monitored function in the memory space;

invoking one of the plurality of functions as a result of receiving a call from an application programming instance;

executing stack walk processing upon the invocation of one of the plurality of functions in the monitored memory space; and

performing, during the executing of the stack walk processing before an address of an originating caller function is reached, a memory check for a plurality of stack entries identified during the stack walk processing to detect suspicious behavior, wherein an alert of suspicious behavior is triggered when the performing of the memory check detects at least one of the following:

code execution is attempted from non-executable memory,

a base pointer is identified as being invalid,

an invalid stack return address is identified,

attempted execution of a return-oriented programming technique is detected,

the base pointer is detected as being outside a current thread stack, and

a return address is detected as being inside a virtual memory area,

wherein when an alert of suspicious behavior is triggered, the method further comprising preventing execution of a payload for the invoked function.

10. The system according to claim **9**, further comprising identifying an address associated with the invoked function; and when the address of the invoked function is found in a

cache, and wherein the processor further executing an operation of preventing execution of a payload for the invoked function.

11. The system according to claim **9**, wherein the executing of the stack walk processing further comprises walking a stack frame associated with the invoked function from a low level user mode function to the originating address of the invoked function.

12. The system according to claim **9**, wherein the monitoring further comprises monitoring functions associated with memory management in the memory space by a user mode function.

13. The system according to claim **9**, wherein the monitoring further comprises applying, upon detecting a function call of one of the plurality of functions in the monitored memory space, a hooking technique to hook a lowest level user mode function of the invoked function, and wherein the executing of the stack walk processing executes a reverse stack walk from the lowest level user mode function of the invoked function.

14. The system according to claim **9**, wherein the plurality of functions comprises an NtAllocateVirtualMemory function, an NtProtectVirtualMemory function, an NtCreateProcess function, an NtCreateProcessEX function, an NtCreateUserIProcess function, and an LdrLoadDll function.

15. The system according to claim **13**, wherein performing of the memory check comprises evaluating state information associated with a cache entry for the hooked function.

16. The system according to claim **15**, wherein when an address of the invoked function is found in a cache, and wherein the processor further executing an operation of preventing execution of a payload for the invoked function.

17. A computer-readable storage device containing instructions, that when executed on at least one processor, causing the processor to execute a process comprising:

monitoring a memory space of a process for execution of at least one monitored function of a plurality of functions, wherein monitoring the memory space comprises loading a component for evaluating the at least one monitored function in the memory space;

invoking one of the plurality of functions as a result of receiving a call from an application programming instance;

executing stack walk processing upon the invocation of one of the plurality of functions in the monitored memory space; and

performing, during the executing of the stack walk processing before an address of an originating caller function is reached, a memory check for a plurality of stack entries identified during the stack walk processing to detect suspicious behavior, wherein an alert of suspicious behavior is triggered when the performing of the memory check detects at least one of the following:

code execution is attempted from non-executable memory,

a base pointer is identified as being invalid,

an invalid stack return address is identified,

attempted execution of a return-oriented programming technique is detected,

the base pointer is detected as being outside a current thread stack,

a return address is detected as being inside a virtual memory area, and

a low level user mode function is attempted to be executed in a cache, wherein the cache is evaluated for an address of the invoked function, and

wherein when an alert of suspicious behavior is triggered, preventing execution of a payload for the invoked function.

18. The computer-readable storage device according to claim **17**, wherein performing of the memory check to determine if code execution is attempted from non-executable memory, the base pointer is identified as being invalid, an invalid stack return address is identified, execution of a return-oriented programming technique is attempted, the base pointer is detected as being outside a current thread stack, and the return address is detected as being inside a virtual memory area, occurs before an address of the invoked function is reached, and

wherein the performing of the memory check to determine whether the low level user mode function is attempted to be executed in the cache occurs upon reaching the address of the invoked function in the execution of the stack walk processing.

* * * * *

Exhibit 6

U 8170010

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

October 22, 2021

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THIS OFFICE OF:

U.S. PATENT: *10,599,844*
ISSUE DATE: *March 24, 2020*

By Authority of the
Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office

M. Tar

M. TARVER
Certifying Officer

US010599844B2

(12) **United States Patent**
Schmidtler et al.

(10) **Patent No.: US 10,599,844 B2**
(45) **Date of Patent: Mar. 24, 2020**

(54) **AUTOMATIC THREAT DETECTION OF EXECUTABLE FILES BASED ON STATIC DATA ANALYSIS**

(71) Applicant: **Webroot Inc.**, Broomfield, CO (US)

(72) Inventors: **Mauritius Schmidtler**, Escondido, CA (US); **Gaurav Dalal**, San Jose, CA (US); **Reza Yoosoofmiya**, San Diego, CA (US)

(73) Assignee: **Webroot, Inc.**, Broomfield, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 95 days.

(21) Appl. No.: **14/709,875**

(22) Filed: **May 12, 2015**

(65) **Prior Publication Data**

US 2016/0335435 A1 Nov. 17, 2016

(51) **Int. Cl.**

G06F 21/56	(2013.01)
G06N 5/04	(2006.01)
G06N 20/00	(2019.01)
G06F 8/53	(2018.01)
G06F 11/30	(2006.01)

(52) **U.S. Cl.**

CPC ***G06F 21/565*** (2013.01); ***G06F 21/562*** (2013.01); ***G06N 20/00*** (2019.01); *G06F 8/53* (2013.01); *G06F 2221/033* (2013.01)

(58) **Field of Classification Search**

CPC .. G06F 21/565; G06F 99/005; G06F 221/033; G06F 21/562; G06F 2221/033; G06F 8/53; G06N 20/00

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

8,510,836	B1 *	8/2013	Nachenberg	H04L 63/145 706/12
8,719,924	B1 *	5/2014	Williamson	G06F 21/568 713/188
9,197,663	B1 *	11/2015	Gilbert	G06F 3/04817
9,465,940	B1 *	10/2016	Wojnowicz	G06F 21/565
2007/0240220	A1 *	10/2007	Tuvell	G06F 21/56 726/24

(Continued)

OTHER PUBLICATIONS

Shabtai, A et al. "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey". Information Security Technical Report, Elsevier Advanced Technology. Amsterdam, NL, vol. 14, No. 1, Feb. 1, 2009, pp. 16-29, XP026144093, ISSN: 1363-4127.*

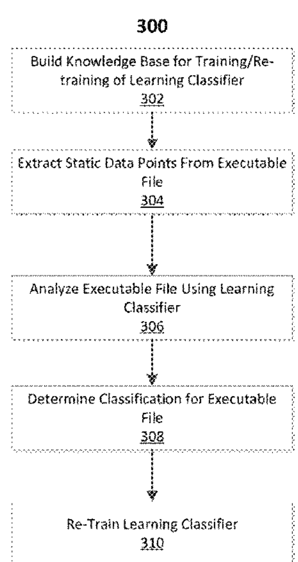
(Continued)

Primary Examiner — Michael Simitoski
(74) *Attorney, Agent, or Firm* — Sprinkle IP Law Group

(57) **ABSTRACT**

Aspects of the present disclosure relate to threat detection of executable files. A plurality of static data points may be extracted from an executable file without decrypting or unpacking the executable file. The executable file may then be analyzed without decrypting or unpacking the executable file. Analysis of the executable file may comprise applying a classifier to the plurality of extracted static data points. The classifier may be trained from data comprising known malicious executable files, known benign executable files and known unwanted executable files. Based upon analysis of the executable file, a determination can be made as to whether the executable file is harmful.

20 Claims, 4 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

2009/0013405	A1 *	1/2009	Schipka	G06F 21/562 726/22
2009/0274376	A1 *	11/2009	Selvaraj	G06K 9/00442 382/224
2009/0300765	A1 *	12/2009	Moskovitch	G06F 21/562 726/24
2010/0082642	A1 *	4/2010	Forman	G06F 16/35 707/749
2010/0192222	A1 *	7/2010	Stokes	G06F 21/563 726/22
2012/0079596	A1 *	3/2012	Thomas	G06F 21/55 726/24
2012/0227105	A1 *	9/2012	Friedrichs	G06F 21/564 726/22
2013/0291111	A1 *	10/2013	Zhou	G06F 21/566 726/23
2014/0165203	A1 *	6/2014	Friedrichs	G06F 21/56 726/24
2014/0310517	A1 *	10/2014	Christodorescu	H04L 41/142 713/160
2016/0021174	A1 *	1/2016	De Los Santos Vilchez	H04L 43/10 709/201
2016/0156646	A1 *	6/2016	Hsueh	G06F 21/563 726/1
2017/0262633	A1 *	9/2017	Miserendino	G06F 21/566
2018/0365573	A1 *	12/2018	Sultana	G06N 5/04

OTHER PUBLICATIONS

Bost, Raphael, et al. "Machine learning classification over encrypted data." NDSS. vol. 4324. 2015. (Year: 2015).*

Choi, Junho, et al. "Efficient malicious code detection using n-gram analysis and SVM." 2011 14th International Conference on Network-Based Information Systems. IEEE, 2011. (Year: 2011).*

Devi, Dhruwajita, and Sukumar Nandi. "Detection of packed malware." Proceedings of the First International Conference on Security of Internet of Things. ACM, 2012. (Year: 2012).*

Dube, Thomas, et al. "Malware target recognition via static heuristics." Computers & Security 31.1 (2012): 137-147. (Year: 2012).*

Merkel, Ronny, et al. "Statistical detection of malicious pe-executables for fast offline analysis." IFIP International Conference on Communications and Multimedia Security. Springer, Berlin, Heidelberg, 2010. (Year: 2010).*

Singla, Sanjam, et al. "A Novel Approach to Malware Detection using Static Classification." International Journal of Computer Science and Information Security 13.3 (2015): 1. (Year: 2015).*

Tabish, S. Momina, M. Zubair Shafiq, and Muddassar Farooq. "Malware detection using statistical analysis of byte-level file content." Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics. ACM, 2009. (Year: 2009).*

Treadwell, Scott, and Mian Zhou. "A heuristic approach for detection of obfuscated malware." 2009 IEEE International Conference on Intelligence and Security Informatics. IEEE, 2009. (Year: 2009).*

International Patent Application No. PCT/US2016/032090, International Search Report and Written Opinion dated Jul. 29, 2016, 13 pages.

Perdisci et al. (2008) Pattern Recognition Letters 29:1941-1946 "Classification of packed executables for accurate computer virus detection".

Perdisci et al. (2008) IEEE: Annual Computer Security Applications Conference 301-310 "McBoost: Boosting Scalability in Malware Collection and Analysis Using Statistical Classification of Executables".

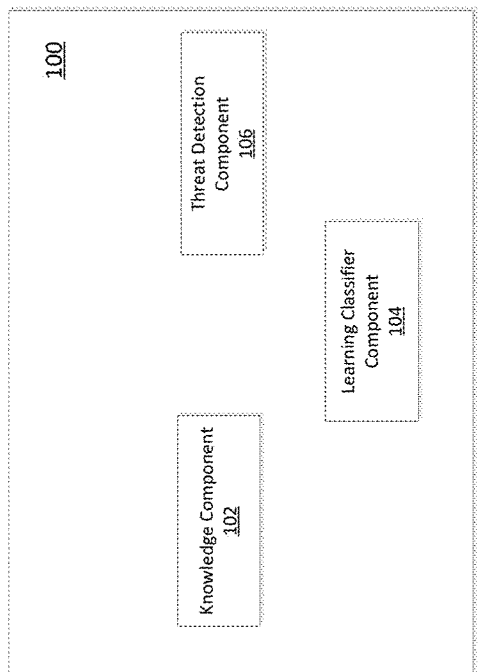
Khan, Hassan et al., "Determining Malicious Executable Distinguishing Attributes and Low-Complexity Detection", J. Comput Virol, 7.2 (2011), pp. 95-105.

Kolter, J. Zico et al., "Learning to Detect and Classift Malicious Executables in the Wild", Journal of Machine Learning Research 7 (2006), pp. 2721-2744.

International Patent Application No. PCT/US2016/032090, International Preliminary Report on Patentability dated Nov. 14, 2017, 7 pages.

* cited by examiner

FIG. 1


100
Threat Detection Component 106
Learning Classifier Component 104
Knowledge Component 102

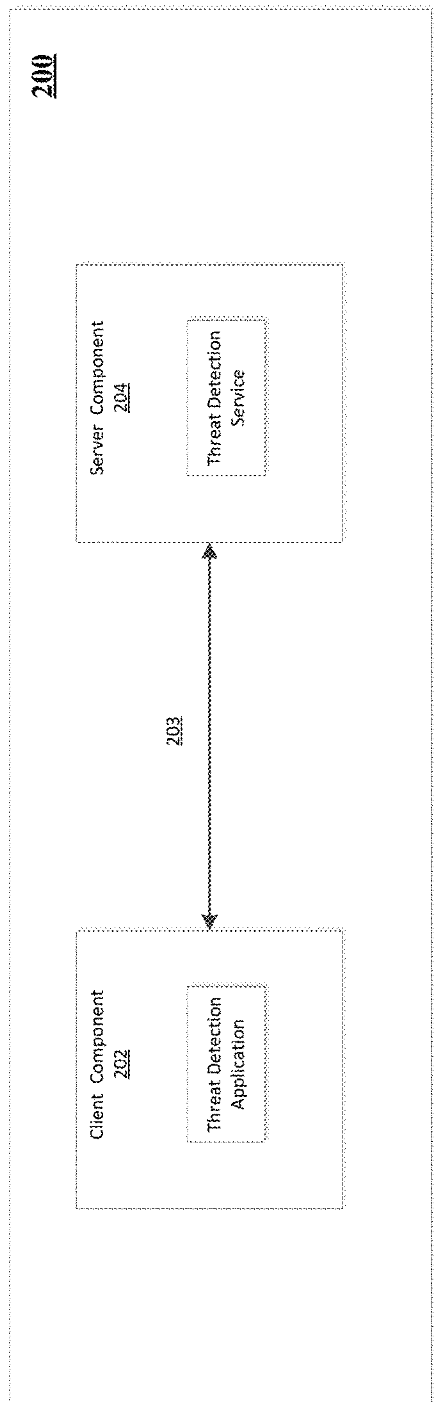
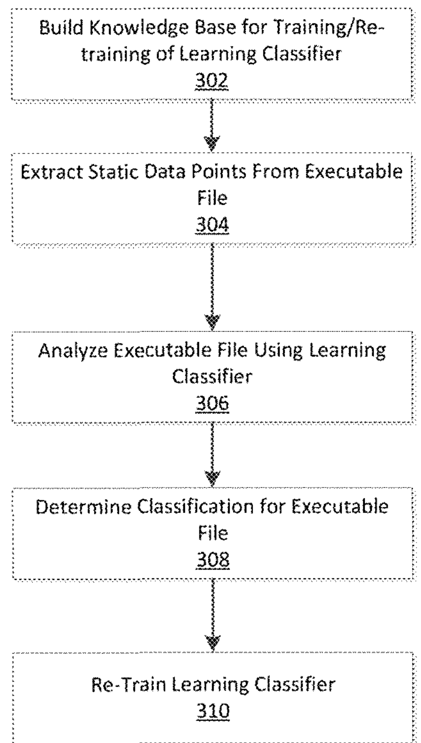

200
Server Component
204
Threat Detection Service
203
Client Component
202
Threat Detection Application

FIG. 2

FIG. 3

300


Build Knowledge Base for Training/Re-training of Learning Classifier
302
Extract Static Data Points From Executable File
304
Analyze Executable File Using Learning Classifier
306
Determine Classification for Executable File
308
Re-Train Learning Classifier
310

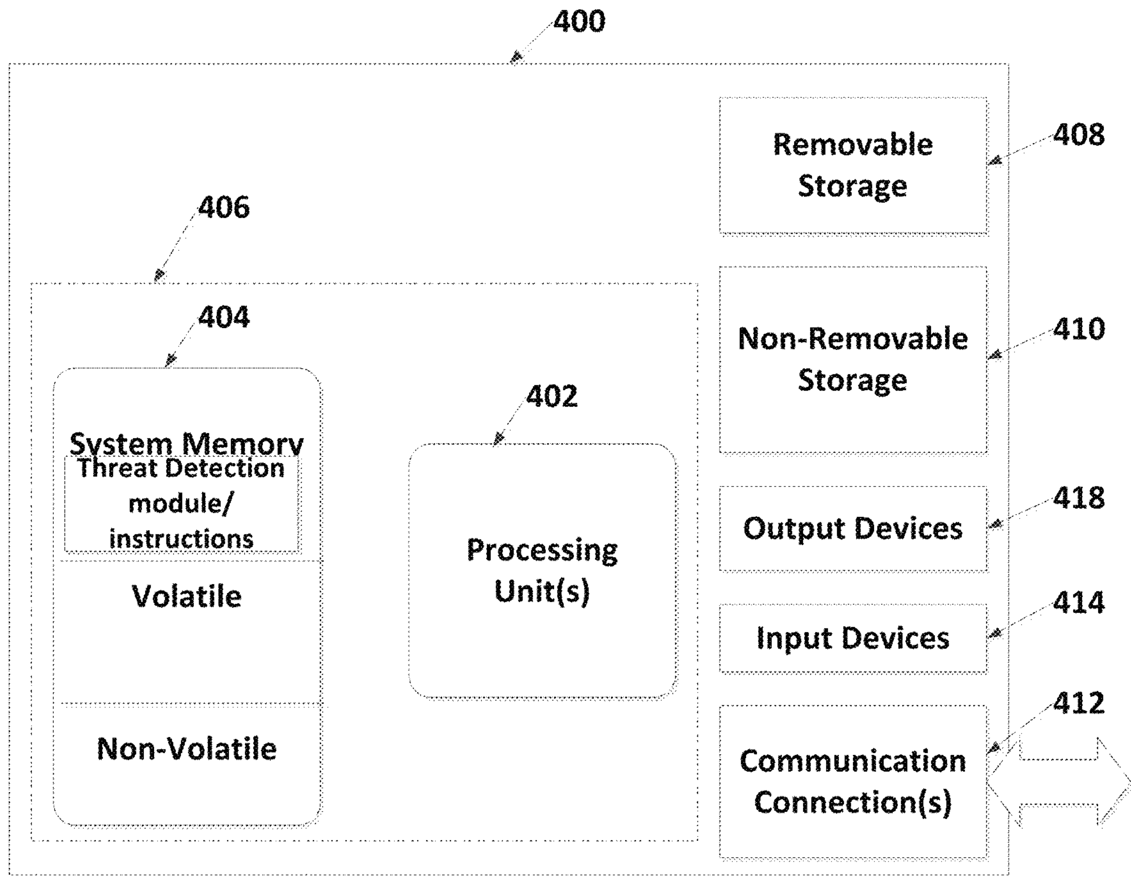


FIG. 4

WBRT007997

AUTOMATIC THREAT DETECTION OF EXECUTABLE FILES BASED ON STATIC DATA ANALYSIS

BACKGROUND

Everyday new executable files are created and distributed across networks. A large portion of these distributed executable files are unknown. For instance, it is not known if such distributed executable files are malicious or not. Given the high volume of new unknown files distributed on a daily basis, it is important to determine threats contained in the set of new unknown files instantaneously and accurately. It is with respect to this general environment that aspects of the present technology disclosed herein have been contemplated.

SUMMARY

Aspects of the present disclosure relate to threat detection of executable files. A plurality of static data points are extracted from an executable file without decrypting or unpacking the executable file. The executable file may then be analyzed without decrypting or unpacking the executable file. Analyzing of the executable file comprises applying a classifier to the plurality of static data points extracted from the executable file. The classifier is trained from data comprising known malicious executable files, known benign executable files and potentially unwanted executable files. Based upon the analysis of the executable file, a determination is made as to whether the executable file is harmful. In some examples, execution of the executable file is prevented when a determined probability value that the executable file is harmful exceeds a threshold value.

This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used to limit the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

Non-limiting and non-exhaustive examples are described with reference to the following figures. As a note, the same number represents the same element or same type of element in all drawings.

FIG. **1** illustrates an exemplary system **100** showing interaction of components for implementation threat detection as described herein.

FIG. **2** illustrates an exemplary distributed system **200** showing interaction of components for implementation of exemplary threat detection as described herein.

FIG. **3** illustrates an exemplary method **300** for implementation of threat detection systems and methods described herein.

FIG. **4** illustrates one example of a suitable operating environment **400** in which one or more of the present examples may be implemented.

DETAILED DESCRIPTION

Non-limiting examples of the present disclosure relate to the threat detection of executable files. The examples disclosed herein may also be employed to detect zero day threats from unknown executable files. However, one skilled in the art will recognize that the present disclosure is not

limited to detection of executable files that may present zero day threats and can be applicable to any unknown file that is attempting to execute on a system (e.g., processing device). The present disclosure is able to detect whether an executable file is harmful or benign before the executable file is actually executed on a processing device. In examples, machine learning processing applies a classifier to evaluate an executable file based on static points data collected from the executable file. The classifier is trained from a collection of data comprising known malicious files, potentially unwanted files and benign files. The classifier is designed and trained such that it can handle encrypted and/or compressed files without decrypting and/or decompressing the files.

Approaches to detecting threats typically focus on finding malicious code blocks within a file and analyzing the behavior of the file. Such approaches are expensive and time-consuming operations that require decrypting encrypted files, dissembling code, analyzing behavior of malware, among other things. Additionally, behavioral detection requires the execution of the potentially malicious code, thereby presenting an opportunity for the malicious code to harm the computing system is executing on. The present disclosure achieves high and accurate classification rates for potentially malicious executables without the need to employ time consuming processing steps like decryption, unpacking or executing unknown files while maintaining a controlled and safe environment. The determination of unknown files is achieved by evaluating static data points extracted from an executable file using a trained classification system that is able to identify potential threats without analyzing executable behavior of a file. The present disclosure also provides for the creation of a training set that adaptively learns what static data points may indicate the existence of malicious code using training data that contains a statistically significant number of both encrypted and non-encrypted examples of malicious or unwanted executable files, among other information. By collecting a large set of training examples as they appear as publicly available and distributed over network connection (e.g., "in the wild"), the present disclosure ensures that its adaptive learning processing is robust enough to comprise sufficient representation of features (and distributions) from files that may be encrypted, not-encrypted, compressed, and uncompressed, among other examples.

A number of technical advantages are achieved based on the present disclosure including, but not limited to: enhanced security protection including automatic detection of threats, reduction or minimization of error rates in identification and marking of suspicious behavior or files (e.g., cut down on the number of false positives), ability to adapt over time to continuously and quickly detect new threats or potentially unwanted files/applications, improved efficiency in detection of malicious files, and improved usability and interaction for users by eliminating the need to continuously check for security threats, among other benefits that will be apparent to one of skill in the art.

FIG. **1** illustrates an exemplary system **100** showing an interaction of components for implementation threat detection as described herein. Exemplary system **100** may be a combination of interdependent components that interact to form an integrated whole for execution of threat detection and/or prevention operations. Components of the systems may be hardware components or software implemented on and/or executed by hardware components of the systems. In examples, system **100** may include any of hardware components (e.g., used to execute/run operating system (OS)),

and software components (e.g., applications, application programming interfaces, modules, virtual machines, runtime libraries, etc.) running on hardware. In one example, an exemplary system **100** may provide an environment for software components to run, obey constraints set for operating, and/or makes use of resources or facilities of the system **100**, where components may be software (e.g., application, program, module, etc.) running on one or more processing devices. For instance, threat detection operations (e.g., application, instructions, modules, etc.) may be run on a processing device such as a computer, a client device (e.g., mobile processing device, laptop, smartphone/phone, tablet, etc.) and/or any other electronic devices, where the components of the system may be executed on the processing device. In other examples, the components of systems disclosed herein may be spread across multiple devices. For instance, files to be evaluated may be present on a client device and information may be processed or accessed from other devices in a network such as, for example, one or more server devices that may be used to perform threat detection processing and/or evaluation a file before execution of the file by the client device.

As one example, system **100** comprises a knowledge component **102**, a learning classifier component **104**, and a threat determination component **106**, each having one or more additional components. The scale of systems such as system **100** may vary and include more or less components than those described in FIG. **1**. In alternative examples of system **100**, interfacing between components of the system **100** may occur remotely, for example where threat detection processing is implemented on a first device (e.g., server) that remotely monitors and controls process flow for threat detection and prevention of a second processing device (e.g., client).

As an example, threat detection may detection exploits that are executable files. However, one skilled in the art will recognize that the descriptions herein referring to executable files are just an example. Threat detection examples described herein can relate to any computer file. In one example, an executable file may be a portable executable (PE) file that is a file format for executables, object code, dynamic link library files (DLLs), and font library files, among other examples. However, one skilled in the art will recognize that executable files are not limited to PE files and can be any file or program that executes a task according to an encoded instruction. Knowledge component **102** described herein may collect data for use in building, training and/or re-training a learning classifier to evaluate executable files. The knowledge component **102** is one or more storages, memories, and/or modules that continuously collects and manages data that may be used to detect threats in files such as executable files. In one example, the knowledge component **102** maintains a robust collection of data comprising known malicious executable files, known benign executable files, and potentially unwanted executable files. As an example, Malicious executable files may be any type of code, data, objects, instructions, etc., that may cause harm or alter an intended function of a system and/or any system resources (e.g., memory, processes, peripherals, etc.) and/or applications running on a device such as an operating system (OS). A benign executable file is a file that upon execution, will not cause harm/damage or alter an intended system function. In other examples, a benign executable may cause harm/damage or alter an intended system; however, the potential harm/damage or alteration may be acceptable to the owner of the system or device. In examples, potentially unwanted executable files are files that may be installed on

system **100** that may not be malicious, but a user of the device may not want such a file to execute and/or a file that is executed/installed unknowingly to a user. In one example, classification of executable files as malicious, benign or potentially unwanted are done by research and development support associated with developments and programming of a threat detection application/module. However, one skilled in the art will recognize that identification and classification of files into the above identified categories may be done by monitoring or evaluating a plurality of resources including but not limited to: network data, executable file libraries and information on previously known malicious executable files as well as benign executable files and potentially unwanted executable files, users/customers of threat detection/computer security software, network flow observed from use of threat detection processing and products, business associations (e.g., other existing threat detection services or partners), third-party feeds, and updates from threat detection performed using learning classifier of present disclosure, among other examples.

To classify executable files into one of the above identified categories, the knowledge component **102** collects static data on a large variety of executable files (e.g., PE files). Examples of different types of executable files collected and evaluated include but are not limited to: bit files (e.g., 32/64 bit files), operating system files (e.g., Windows, Apple, Linux, Unix, etc.), custom built files (e.g., internal tool files), corrupted files, partial downloaded files, packed files, encrypted files, obfuscated files, third party driver files, manually manipulated binary files, unicode files, infected files, and/or memory snapshots, among other examples. The data collected and maintained by the knowledge component **102** yields a knowledgebase that may be used to periodically train a classifier, e.g. learning classifier component **104** utilized by system **100**. The learning classifier component may be used to classify an executable file as one of the following classifications: malicious, benign or potentially unwanted. In some examples, classification may span two or more two or more of those classification categories. For example, an executable file may be benign in the sense that it is not harmful to system **100** but may also be classified as potentially unwanted as it might be installed without explicit user consent, for example. Data maintained by the knowledge component **102** may be continuously updated system **100** or a service that updates system **100** with new exploits to add to the training sample. For example, a research team may be employed to continuously collect new examples of harmful executables, benign executables, and potentially unwanted executables, as many unknown executable files are generated on a daily basis over the Internet. Executable files may be evaluated by the research team such as by applying applications or processing to evaluate executable files including data associated with the file and/or actions associated with the file (e.g., how it is installed and what a file does upon execution). Continuous update of the data maintained by the knowledge component **102** in conjunction with on-going re-learning/re-training of the learning classifier component **104** ensures that system **100** is up-to-date on current threats. Knowledge of the most current threats improves the generalization capability of system **100** to new unknown threats. By incorporating malicious files, benign files and potentially unwanted files, the present disclosure greatly improves a knowledge base that may be used to in training a learning classifier thereby resulting in more accurate classifications as compared with other knowledge bases that are based on only malicious and/or benign files.

In examples, the collected data on the executable files is analyzed to identify static data points that may indicate one of a malicious file, a benign file or a potentially unwanted file. For instance, the knowledge component **102** may employ one or more programming operations to identify static data points for collection, and to associate the static data points with one of the categories of files (e.g., malicious, benign or potentially unwanted). Programming operations utilized by the knowledge component **102** include operations to collect file data (e.g., executable files or data points from executable files), parse the file data, and store extracted data points. In at least one example, the knowledge component **102** comprises one or more components to manage file data. For example, the knowledge component **102** may comprise one or more storages such as databases, and one or more additional components (e.g., processors executing programs, applications, application programming interfaces (APIs), etc.).

Further, the knowledge component **102** may be configured to continuously collect data, generate a more robust collection of file data to improve classification of file data and train a classifier used to detect whether an executable file is harmful, benign, or potentially unwanted. The identification of static data points to be collected and analyzed for executable files may be continuously updated as more information becomes available to the knowledge component **102**. A static data point may be a point of reference used to evaluate an executable file. As examples, static data points include, but are not limited to: header information, section information, import and export information, certificate information, resource information, string and flag information, legal information, comments, and/or program information

(e.g., APIs and DLLs), among other examples. Static data points may be organized into categories that identify a type of static data point. Categories of static data points comprise, but are not limited to: numeric values, nominal values string sequences, byte sequences, and/or Boolean values, among other examples. Any number of static data points may be collected and analyzed for an executable file. In general, collecting and analyzing a greater number of static data points results in more accurate classification of an executable file. For example, eighty static data points may be identified and collected (or attempted to be collected) from an executable file during analysis of an executable file. While a specific number is static data points are provided herein, one of skill in the art will appreciate the more or fewer static data points may be collected without departing from the scope of this disclosure. As an example, the following table, Table 1.1, identifies some of the static data points identified for analysis of an executable file, where the static data points are organized by category:

TABLE 1.1

Numeric values	Nominal values	Strings/Byte sequences	Boolean values
File size	initialize	Comments	Address Of Entry Point Anomaly
linker version	Un-initialize	company name	Image Base Anomaly
code size	entry point	file description	Section Alignment Anomaly
OS version	subsystem	internal name	Size Of Code Mismatch Anomaly
image version	file subtype	legal copyright	Low Import Count Anomaly
subsystem version	language	original file	Entry Point Anomaly
file version number	file flags masks	private build	certificate Validity
product version number	file flags	product name	Certificate Exception
size of heapr	file OS	special build	Code Characteristics Anomaly
size of stackr	file type	product version	Code Name Anomaly
size of image	machine type	file version	Count Anomaly
PE header time	PE type	package code	Data Characteristics Anomaly
Section Entropy	section counts	product code	Data Name Anomaly
Sections count	DLL count	export DLL name	Export Exception
	DLL functions	assembly version	Large Number of DLLs Anomaly
	data directory	Certificate Issuer	flag DLL Name Anomaly
	export count	Certificate Subject	Number of Functions Anomaly
	Earliest Data Byte	Imports	Function Name Anomaly
	resources	Exports	PE Header Anomaly
	resources language	Section Names	High Section Count Anomaly
	resource Encoding	Non-resource section strings	PE Magic Validity
	resource code page		Resource Exception
	resource size		VR Code Ratio Anomaly
	DLL characteristics		Import Exception

Examples of the present disclosure need not distinguish between encrypted files and non-encrypted files. The static data points may be extracted from files regardless of whether or not they are encrypted and/or compressed. Given that the training set contains a statistically significant number of encrypted and non-encrypted examples, as well as compressed and decompressed files, a learning classifier (e.g., learning classifier component **104**) may be trained to identify features in the extracted data to identify malicious files. Since a very large majority of the files "in the wild" use few

tools to encrypt the files (e.g., encryption algorithm, packers, etc.), the distribution of the data found across large number of files is preserved after encryption, although the actual content of the data is transformed into something different. By collecting a large set of training examples as they appear "in the wild", a sufficient representation of features and distributions associated with the represented features exist in a training set that are both encrypted and not-encrypted. The present disclosure provides at least one benefit over other threat detection applications/programs by utilizing a very large and diverse training sample, thereby enabling a more intelligent and adaptive learning classifier that is able to achieve high detection rates for threats and low false positive rates, among other benefits.

In addition to collecting and managing information to train a learning classifier, the knowledge component **102** is used to evaluate an executable file and extract/collect static data points for evaluation of the file by the learning classifier before the file is executed. In one example, the system automatically identifies a new file (e.g., unknown file) for evaluation. As examples, identification of an executable file for evaluation may occur in examples such as: when a download of a file is requested, while a download is being performed, evaluation of streaming data, when a new file is detected as attempting to execute (e.g., potentially unwanted file), and before an unknown file or file containing executable code that was not previously checked attempts to execute, among other examples. In another example, a user of a device with which components of system **100** are operating may identify a file to be evaluated. The knowledge component **102** collects as many static data points as it can to evaluate the executable file using a learning classifier built by the learning classifier component **102**. In an exemplary extraction, the knowledge component **102** extracts each of the static data points identified in Table 1.1. The learning classifier component **102** intelligently builds a learning classifier to evaluate a file based on the extracted static data points of the file and the data managed by the knowledge component **102**.

The learning classifier component **104** is a component used to evaluate an executable file using the information provided by the knowledge component **102**. The learning classifier component **104** interfaces with the knowledge component **102** and a threat detection component **106** for the system **100** to evaluate an executable file as a possible threat. As an example, the learning classifier **104** applies programming operations or machine learning processing to evaluate static data points extracted from a file to be analyzed. In doing so, the learning classifier component **104** builds a learning classifier based on information from the knowledge component **102** including the extracted static data points of a file and the information used to train/re-train the learning classifier (e.g., static data on the robust collection of executable files including variety of file types of malicious executable files, benign executable files, and potentially unwanted executable files).

As an example, the learning classifier can adaptively set features of a learning classifier based on the static data points extracted for a file. For example, ranges of data can be identified for static data points that may enable the learning classifier to controllably select (e.g., turn on/off) features for evaluation by learning classifier. In one instance where a static data point extracted for evaluation is a file size (e.g., numeric value), the learning classifier may turn on/off features for evaluation based on whether a the file size of the file is within a certain range. In another example, the learning classifier may detect that the file does not contain "legal information" in the file. Legal information may be any identifying information indicating data that conveys rights to one or more parties including: timestamp data, licensing information, copyright information, indication of intellectual property protection, etc. In an example where the learning classifier detects the presence of legal information does not exist in the executable file, this may trigger the learning classifier to adaptively check for additional features that might be indicative of a threat or malicious file as well as turn off other features related to an evaluation of the "legal information." The learning classifier component **104** utilizes programming operations or machine-learning processing to train/re-train the learning classifier to uniquely evaluate any file/file type. One of skill in the art will appreciate different types of processing operations may be employed without departing from the spirit of this disclosure. For example, the learning classifier component **104** may utilize an artificial neural network (ANN), a decision tree, association rules, inductive logic, a support vector machine, clustering analysis, and Bayesian networks, among other examples.

The learning classifier component **104** processes and encodes collected/extracted data from a file to make the static data points suitable for processing operations. The processing and encoding executed by the learning classifier component **104** may vary depending on the identified categories and/or type (e.g., numeric values, nominal values, string/byte sequences, Boolean values, etc.) of static data points collected by the knowledge component **102**. As an example, string sequence data as well as byte sequence data may be parsed and processed as n-grams and/or n-gram word prediction (e.g., word-grams). For instance, for a given string all unigrams, bigrams and so forth up to a given length n are generated and the counts of the individual n-grams are determined. The resulting counts of the unique n-grams string and/or byte sequences are then used as input to a generated feature vector. In one example, strings are processed directly according to a bag of word model. A bag of word model is a simplifying representation used in natural language processing and information retrieval. In another example, numeric value static data points may be binned appropriately ensuring a good balance between information loss through data binning and available statistics. Nominal values as well Boolean values may be encoded using true/false flags. All of the different encoded data may be combined into one or more final feature vectors, for example after a complex vector coding/processing is performed (e.g., L2 norm).

As an application example of processing performed by the learning classifier component **104**, suppose extraction of static data points from a file identifies the following four (4) data points:

File size: 4982784

PEHeader Anomaly: False

Section name: .TEXT

Legal copyright: Copyright (C) Webroot Inc. 1997

Processing these 4 data points into a feature vector may require binning the file size, labeling the PE Header Anomaly, build n-grams for the section name and building word-gram for legal copyright. Encapsulation of special characters that sometimes appear in text (e.g., section names) may be desirable. In an example, strings of data may be transformed first into hex code representation and then build n-grams or word-grams. In this particular example, the section name is transformed into n-grams while the legal copyright text is transformed into word-grams. In evaluation using the classifier, the features related to the extracted data

points may be weighted by the particular data point being evaluated or by category, for example. For instance, the training of the learning classifier may indicate that legal information (e.g., "legal copyright") provides a better indication that a file may be malicious than that of a section name. Features being evaluated (and their distributions) differ for benign files as compared to malicious files (e.g., harmful executables such as malware). For example, the data field "Legal Copyright" for benign files tends to have meaningful words, whereas in a malicious file, this data field is often left empty or it is filled with random characters. Commercial software files tend to have a valid certificate, whereas malware in general do not have valid certificates. Similarly, each data field for a static data point evaluated provides further indication about the maliciousness of the file. The combined information from all these data fields coupled with machine learning enables accurate prediction determination as to whether a file is malicious, benign or potentially unwanted. As an example, the resulting feature vector in sparse form is shown below:

secname_002e00740065:0.204124145232
secname_0065:0.0944911182523
secname_0074006500780074:0.353553390593
secname_006500780074:0.408248290464
secname_00740065:0.144337567297
secname_002e007400650078:0.353553390593
secname_00650078:0.144337567297
secname_002e0074:0.144337567297
secname_002e:0.0944911182523
secname_00780074:0.433012701892
secname_0078:0.0944911182523
secname_007400650078:0.204124145232
secname_0074:0.472455591262
legalcopyright_0043006f0070007900720069006700-80074:0.4472135955
legalcopyright_002800430029:0.4472135955
legalcopyright_0043006f00720070002e:0.4472135955
legalcopyright_0031003900390035:0.4472135955
legalcopyright_004d006900630072006f0073006f00-660074:0.4472135955
filesize_9965:1.0
PEHeaderAnomaly_False:1.0

In examples, the learning classifier component **104** provides the one or more feature vectors as input to a support vector machine (SVM). The SVM may then perform data analysis and pattern recognition on the one or more feature vectors. In one example the SVM, may be a linear SVM. Given a set of training examples provided by the knowledge component **102**, an SVM may build a probabilistic model that indicates whether or not a file may be malicious. In another example, a hybrid approach may be elected that combines two or more individual linear SVM classifiers into a final classifier using ensemble methods. Specifically, the evaluated static data points may be subdivided into a set of families (e.g., sections, certificate, header data, and bytes sequences are used as different families). For each family, a linear SVM may be trained. The resulting classification scores generated by each linear SVM may then be combined into a final classification score using a decision tree. As an example, the decision tree may be trained using two-class logistic gradient boosting. In yet another example of feature classification evaluation, a classification may be subdivided into a three class classification problem defined by malicious files, potentially unwanted files/applications and benign files. The resulting three class problem is solved using multi-class classification (e.g., Directed Acyclic Graph (DAG) SVM) or, in case of the hybrid approach based on feature families, using a decision tree based on three-class logistic gradient boosting.

The threat detection component **106** is a component of system **100** that evaluates results of the processing performed by the learning classifier component **104** to make a determination as to whether the executable file is malicious, benign or a potentially unwanted file/application. As an example, a probabilistic value may be a final score determined from evaluating all static data points (or feature distributions) individually and determining an aggregate score that may be used for the evaluation of an executable file. In another example, correlation between static data points may be determined by the learning classification component **104** and a final score may be generated based on the correlation between static data points evaluated.

In one example, a classification as to whether a file is malicious or not may be based on comparison with a predetermined threshold value. For instance, a final score (e.g., probability value/evaluation) is determined based on feature vector processing performed by the learning classifier component **104** and compared with a probability threshold value for determining whether an executable file is malicious. As an example, a threshold value(s) may be set based on predetermined false positive range data where ranges may be set for one or more of the malicious executable files, the benign executable files and the potentially unwanted executable files. The threshold values for each of the different types may be the same or different. For instance, a threshold value may be set that indicates a confidence score in classifying an executable file as a malicious executable file. Ranges may be determined indicating how confident the system **100** is in predicting that a file is malicious. That may provide an indication of whether further evaluation of an executable file should occur. However, one skilled in the art will recognize that threshold determinations can be set in any way that be used to determine whether an executable file is malicious or not. Examples of further evaluation include but are not limited to: additional processing using a learning classifier, identification of an executable file as potentially malicious where services associated with the system **100** may follow-up, quarantining a file, and moving the file to a secure environment for execution (e.g., sandbox) among other examples. In other examples, retraining of the classifier may occur based on confidence scores. In examples, when a probability value exceeds or threshold value (or alternatively is less than a threshold value), it may be determined that an executable file may be identified as malicious. One skilled in the art will recognize that determination for predictive classification of executable files is not limited to threshold determinations. For example, any type of analytical, statistical or graphical analysis may be performed on data to classify an executable file/unknown executable file. As identified above, the threat detection component **106** may interface with the learning classifier component **104** to make a final determination as to how to classify an executable file as well as interface with the knowledge component **102** for training/retraining associated with the learning classifier.

FIG. **2** illustrates an exemplary distributed system **200** showing interaction of components for implementation of exemplary threat detection system as described herein. Where FIG. **1** illustrates an example system **100** having components (hardware or software) operating on a client, FIG. **2** illustrates an exemplary distributed system **200** comprising a client component **202** connected with at least one server component **204** via communication line **203**. Communication line **203** represents the ability of the client component **202** to communicate with the server component **204**, for example, to send or receive information over a

network connection. That is, client component **202** and server component **204** are connectable over a network connection (e.g., a connection Internet via, for example, a wireless connection, a mobile connection, a hotspot, a broadband connection, a dial-up, a digital subscriber line, a satellite connection, an integrated services digital network, etc.).

The client component **202** may be any hardware (e.g., processing device) or software (e.g., application/service or remote connection running on a processing device) that accesses a service made available by the server component **204**. The server component **204** may be any hardware or software (e.g., application or service running on a processing device) capable of communicating with the client component **202** for execution of threat detection processing (e.g., threat detection application/service). Threat detection processing may be used to evaluate a file before execution of the file as described in FIG. **1**. Threat detection application or services may be present on at least one of the client component **202** and the server component **204**. In other examples, applications or components (e.g., hardware or software) may be present on both the client component **202** and the server component **204** to enable processing by threat detection application/services when a network connection to the server cannot be established. In one example of system **200**, client component **202** (or server component **204**) may comprise one or more components for threat detection as described in system **100** including a knowledge component **102**, a learning classifier component **104** and/or a threat detection component **106**, as described in the description of FIG. **1**. In other examples, the client component **202** may transmit data to/from the server component **204** to enable threat detections services over distributed network **200**, for example as represented by communication line **203**. In one example, threat detection applications/services operating on the client component **202** may receive updates from the server component **204**. For instance, updates may be received by the client component **202** for re-training of a learning classifier used to evaluate an executable file on the client component **202**.

FIG. **3** illustrates an exemplary method **300** for performing threat detection. As an example, method **300** may be executed by an exemplary system such as system **100** of FIG. **1** and system **200** of FIG. **2**. In other examples, method **300** may be executed on a device comprising at least one processor configured to store and execute operations, programs or instructions. However, method **300** is not limited to such examples. Method **300** may be performed by any application or service that may include implementation of threat detection processing as described herein. The method **300** may be implemented using software, hardware or a combination of software and hardware.

Method **300** begins at operation **302** where a knowledge base is built for training/retraining of a learning classifier used to detect threats in executable files. Operation **302** builds the knowledge base from data collected and evaluated related to known malicious executable files, known benign executable files and potentially unwanted executable files as described in the description of FIG. **1** (e.g., knowledge component **102**). The knowledge base may be used to automatically train/re-train one or more learning classifiers used to evaluate threats in executable files based on the known malicious executable files, known benign executable files and the potentially unwanted executable files. In examples, the knowledge base may be maintained on at least one of a client component and a server component, and the knowledge base may be continuously updated with information from the resources described in FIG. **1** including update information based on threat detection processing evaluation performed on unknown executable files.

When an executable file is identified for evaluation, flow proceeds to operation **304** where a one or more static data points are extracted from an executable file. In operation **304**, an executable file is analyzed using machine learning processing as described with respect to the knowledge component **102** of FIG. **1** to collect/extract static data points from an executable file for evaluation. In examples, operation **304** occurs without decrypting or unpacking the executable file. However, in other examples, machine learning processing performed has the capability to evaluate static data points from extracted/unpacked content. In one example, machine learning processing is used to extract static data points from encrypted and/or compress versions of one or more files. In another example, machine learning processing is used to extract static data points from decrypted and/or unpacked versions of one or more files. In any example, extraction of static data points from different files (e.g. executable files) can be used to enhance training of a learning classifier, providing better results for extraction of static data points and classification of files.

In examples, operation **304** further comprises classifying extracted data according to a type of data extracted. For instance, the actions performed at operation **304** may be used to classify the plurality of static data points extracted into categories (categorical values) comprising numeric values, nominal values, string or byte sequences, and Boolean values, for example, as described with respect to FIG. **1**. In examples, data of an executable file (e.g., binary file data) may be parsed and compared against data maintained by a threat detection application/service as described in the present disclosure to determine static data points for extraction/collection.

In operation **306**, the executable file is analyzed for threats. As an example, operation **306** analyzes an executable without decrypting or unpacking the executable file. However, in other examples, the machine learning processing being performed has the capability to evaluate static data points from extracted/unpacked content. Operation **306** comprises applying a learning classifier (e.g., the learning classifier generated during performance of operation **302**) to the plurality of static data points extracted from the file. As discussed, the learning classifier may be built from data comprising known malicious executable files, known benign executable files and known unwanted executable files, for example. In one example, operation **306** comprises generating at least one feature vector from the plurality of static data points extracted using the learning classifier trained by the knowledge base. In order to generate the feature vector for the learning classifier, data may be parsed and encoded for machine learning processing.

In one example, generation of the feature vector may comprise selectively setting features of the learning classifier based on the one or more of static data points extracted. Features of the generated feature vector may be weighted based on classified categories identified by the knowledge base (as described above) and the plurality of static data points extracted from the file. As an example, one or more features of the feature vector may be selectively turned on or off based on evaluation of whether a value of a static data point is within a predetermined range. However, one skilled in the art will recognize that the learning classifier can uniquely generate a feature vector for analysis of the executable file based on any data used to train/ re-train the learning classifier. In examples, operation **306** further comprises

evaluating the feature vector using linear or nonlinear support vector processing to determine a classification for the executable file, for example whether the executable file is harmful, benign, or unwanted.

Flow proceeds to operation **308**, where a determination is made as to a classification of the executable file. For example, operation **308** makes a final determination as to whether the executable file is harmful (e.g., malicious, malware) or not based on results of the analysis of the executable file (e.g., using machine learning processing by a learning classifier). In one example, results of the analysis of an executable file may be data obtained from learning classifier, such as, for example an SVM, processing data. In one example, operation **308** further comprises preventing execution of the executable file when a probability value that the executable file is harmful exceeds a threshold value. The probability value for the executable file may be determined based on applying the learning classifier to the executable file. As an example, the threshold value may be set based on predetermined false positive range data for identifying a malicious or harmful executable file. False positive range data may be determined from the analysis/evaluation of the known malicious executable files, known benign files and potentially unwanted executable files/applications, of the knowledge base. However, as acknowledged above, determining a classification of an executable file may be based on any type of analytical, statistical or graphical analysis, or machine learning processing. In one example, ranges can be based on evaluation of data during operation of the threat detection service as well as analysis related to unknown files, for example analytics performed to evaluate unknown files.

At any point in time, operation **310** may occur where a learning classifier used for threat detection processing is re-trained. Continuously re-training of the learning classifier may ensure that the threat detection application/service is up to date and able to accurately detect new threats. As identified above, re-training may occur through results of threat detection processing including updated information added to the knowledge base. In one example, training of a learning classifier can be based on evaluation of data during operation of the threat detection service as well as analysis related to unknown files, for example analytics performed to evaluate unknown files.

FIG. **4** and the additional discussion in the present specification are intended to provide a brief general description of a suitable computing environment in which the present invention and/or portions thereof may be implemented. Although not required, the embodiments described herein may be implemented as computer-executable instructions, such as by program modules, being executed by a computer, such as a client workstation or a server. Generally, program modules include routines, programs, objects, components, data structures and the like that perform particular tasks or implement particular abstract data types. Moreover, it should be appreciated that the invention and/or portions thereof may be practiced with other computer system configurations, including hand-held devices, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

FIG. **4** illustrates one example of a suitable operating environment **400** in which one or more of the present embodiments may be implemented. This is only one example of a suitable operating environment and is not intended to suggest any limitation as to the scope of use or functionality. Other well-known computing systems, environments, and/or configurations that may be suitable for use include, but are not limited to, personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, programmable consumer electronics such as smart phones, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

In its most basic configuration, operating environment **400** typically includes at least one processing unit **402** and memory **404**. Depending on the exact configuration and type of computing device, memory **404** (storing, among other things, executable evaluation module(s), e.g., malware detection applications, APIs, programs etc. and/or other components or instructions to implement or perform the system and methods disclosed herein, etc.) may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.), or some combination of the two. This most basic configuration is illustrated in FIG. **4** by dashed line **406**. Further, environment **400** may also include storage devices (removable, **408**, and/or non-removable, **410**) including, but not limited to, magnetic or optical disks or tape. Similarly, environment **400** may also have input device(s) **414** such as keyboard, mouse, pen, voice input, etc. and/or output device(s) **416** such as a display, speakers, printer, etc. Also included in the environment may be one or more communication connections, **412**, such as LAN, WAN, point to point, etc.

Operating environment **400** typically includes at least some form of computer readable media. Computer readable media can be any available media that can be accessed by processing unit **402** or other devices comprising the operating environment. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media includes volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other non-transitory medium which can be used to store the desired information. Computer storage media does not include communication media.

Communication media embodies computer readable instructions, data structures, program modules, or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of the any of the above should also be included within the scope of computer readable media.

The operating environment **400** may be a single computer operating in a networked environment using logical connec-

tions to one or more remote computers. The remote computer may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above as well as others not so mentioned. The logical connections may include any method supported by available communications media. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

The different aspects described herein may be employed using software, hardware, or a combination of software and hardware to implement and perform the systems and methods disclosed herein. Although specific devices have been recited throughout the disclosure as performing specific functions, one of skill in the art will appreciate that these devices are provided for illustrative purposes, and other devices may be employed to perform the functionality disclosed herein without departing from the scope of the disclosure.

As stated above, a number of program modules and data files may be stored in the system memory **404**. While executing on the processing unit **402**, program modules **408** (e.g., applications, Input/Output (I/O) management, and other utilities) may perform processes including, but not limited to, one or more of the stages of the operational methods described herein such as method **300** illustrated in FIG. **3**, for example.

Furthermore, examples of the invention may be practiced in an electrical circuit comprising discrete electronic elements, packaged or integrated electronic chips containing logic gates, a circuit utilizing a microprocessor, or on a single chip containing electronic elements or microprocessors. For example, examples of the invention may be practiced via a system-on-a-chip (SOC) where each or many of the components illustrated in FIG. **4** may be integrated onto a single integrated circuit. Such an SOC device may include one or more processing units, graphics units, communications units, system virtualization units and various application functionality all of which are integrated (or "burned") onto the chip substrate as a single integrated circuit. When operating via an SOC, the functionality described herein may be operated via application-specific logic integrated with other components of the operating environment **400** on the single integrated circuit (chip). Examples of the present disclosure may also be practiced using other technologies capable of performing logical operations such as, for example, AND, OR, and NOT, including but not limited to mechanical, optical, fluidic, and quantum technologies. In addition, examples of the invention may be practiced within a general purpose computer or in any other circuits or systems.

This disclosure described some aspects of the present technology with reference to the accompanying drawings, in which only some of the possible embodiments were shown. Other aspects may, however, be embodied in many different forms and should not be construed as limited to the embodiments set forth herein. Rather, these aspects were provided so that this disclosure was thorough and complete and fully conveyed the scope of the possible embodiments to those skilled in the art.

Although specific aspects were described herein, the scope of the technology is not limited to those specific embodiments. One skilled in the art will recognize other embodiments or improvements that are within the scope and spirit of the present technology. Therefore, the specific structure, acts, or media are disclosed only as illustrative embodiments. The scope of the technology is defined by the following claims and any equivalents therein.

What is claimed is:

1. A computer-implemented method comprising:

extracting a plurality of static data points from an executable file without decrypting or unpacking the executable file, wherein the plurality of static data points represent predefined character strings in the executable file;

generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points based on a collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files, wherein the collection of data comprises at least a portion of the plurality of static data points, and wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more static data points from the plurality of extracted static data points is within a predetermined range; and

evaluating the feature vector using support vector processing to determine whether the executable file is harmful.

2. The computer-implemented method according to claim **1**, wherein the extracting the plurality of static data points further comprises classifying the extracted static data points according to a type of data extracted from the executable file, and encoding the extracted static data points for the classifier based on the classification.

3. The computer-implemented method according to claim **2**, wherein generating the at least one feature vector comprises selectively setting features of the classifier based on the plurality of extracted static data points.

4. The computer-implemented method according to claim **2**, wherein the classifying the at least one static data point comprises classifying the plurality of extracted static data points into categories comprising numeric values, nominal values, string or byte sequences, and Boolean values.

5. The computer-implemented method according to claim **4**, wherein analyzing the executable file further comprises generating at least one feature vector from the plurality of extracted static data points, wherein features of the generated feature vector are weighted based at least upon the classified categories and the plurality of extracted static data points.

6. The computer-implemented method according to claim **1**, wherein determining whether the executable file is harmful further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.

7. The computer-implemented method according to claim **6**, wherein the threshold value is set based on predetermined false positive range data.

8. The computer-implemented method according to claim **1**, wherein the plurality of static data points is extracted using a machine learning technique.

9. The computer-implemented method according to claim **1**, wherein a determination of whether the executable file is harmful is used to retrain the classifier.

10. A computer-readable storage device containing instructions, that when executed on at least one processor, causing the processor to execute a process comprising:

extracting a plurality of static data points from an executable file without decrypting or unpacking the execut-

able file, wherein the plurality of static data points represent predefined character strings in the executable file;

generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points based on a collection of data comprising known malicious executable files, known benign executable files and known unwanted executable files, wherein the collection of data comprises at least a portion of the plurality of static data points, and wherein one or more features of the feature vector are selectively turned on or off based on whether one or more values of one or more static data points from the plurality of extracted static data points is within a predetermined range; and

evaluating the feature vector using support vector processing to determine whether the executable file is harmful.

11. The computer-readable storage device according to claim **10**, wherein the extracting of the plurality of static data points executed by the processor further comprises classifying the extracted static data points according to a type of data extracted from the executable file, and encoding the extracted static data points for the classifier based on the classifying.

12. The computer-readable storage device according to claim **11**, wherein generating the at least one feature vector comprises selectively setting features of the classifier based on the plurality of extracted static data points.

13. The computer-readable storage device according to claim **11**, wherein the classifying the at least one static data point comprises classifying the plurality of extracted static data points into categories comprising numeric values, nominal values, string or byte sequences, and Boolean values.

14. The computer-readable storage device according to claim **13**, wherein analyzing the executable file further comprises generating a feature vector from the plurality of extracted static data points, wherein features of the generated feature vector are weighted based at least upon the classified categories and the plurality of extracted static data points.

15. The computer-readable storage device according to claim **10**, wherein determining whether the executable file is harmful further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.

16. The computer-readable storage device according to claim **15**, wherein the threshold value is set based on predetermined false positive range data.

17. A system comprising:

at least one memory; and

at least one processor operatively connected with the memory and configured to perform operation of:

extracting a plurality of predefined character strings from an executable file without decrypting or unpacking the executable file;

generating a feature vector from the plurality of predefined character strings using a classifier trained to classify the plurality of predefined character strings based on a collection of data comprising known malicious executable files, known benign executable files and known unwanted executable files, wherein the collection of data comprises at least a portion of one or more of the plurality of predefined character strings, and wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more predefined character strings from the plurality of predefined character strings is within a predetermined range; and

evaluating the feature vector using support vector processing to determine whether the executable file is harmful.

18. The system according to claim **17**, wherein the determining further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value, and wherein the threshold value is set based on predetermined false positive range data.

19. The system according to claim **17**, wherein the extracting the plurality of predefined character strings further comprises classifying the extracted plurality of predefined character strings according to a type of data extracted from the executable file, and encoding the extracted plurality of predefined character strings for the classifier based on the classification.

20. The system according to claim **17**, wherein determining whether the executable file is harmful further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.

* * * * *

Exhibit 7

U 8164950

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

October 07, 2021

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THIS OFFICE OF:

U.S. PATENT: *9,413,721*

ISSUE DATE: *August 09, 2016*

By Authority of the

Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office

R GLOVER

Certifying Officer

US009413721B2

(12) **United States Patent**
Morris et al.

(10) **Patent No.:** **US 9,413,721 B2**
(45) **Date of Patent:** **Aug. 9, 2016**

(54) **METHODS AND APPARATUS FOR DEALING WITH MALWARE**

(75) Inventors: **Melvyn Morris**, Belper (GB); **Joseph Jaroch**, Elk Grove Village, IL (US)

(73) Assignee: **WEBROOT INC.**, Broomfield, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **13/372,375**

(22) Filed: **Feb. 13, 2012**

(65) **Prior Publication Data**

US 2012/0260340 A1 Oct. 11, 2012

Related U.S. Application Data

(60) Provisional application No. 61/443,095, filed on Feb. 15, 2011.

(51) **Int. Cl.**
H04L 29/06 (2006.01)
G06F 21/56 (2013.01)

(52) **U.S. Cl.**
CPC ***H04L 63/0263*** (2013.01); ***G06F 21/56*** (2013.01); ***G06F 21/566*** (2013.01); ***H04L 63/14*** (2013.01); ***H04L 63/145*** (2013.01)

(58) **Field of Classification Search**
USPC 726/23
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,175,869 B1 1/2001 Ahuja et al.
6,219,786 B1 4/2001 Cunningham et al.
6,842,878 B1 1/2005 Johnson et al.
7,051,365 B1 5/2006 Bellovin
7,188,173 B2 * 3/2007 Anderson et al. 709/225
7,231,440 B1 * 6/2007 Kouznetsov et al. 709/224
7,343,626 B1 3/2008 Gallagher
7,555,476 B2 6/2009 Holbrook
7,707,634 B2 4/2010 Sandu et al.
7,761,912 B2 7/2010 Yee et al.
7,814,056 B2 * 10/2010 McGrattan et al. 707/640

(Continued)

FOREIGN PATENT DOCUMENTS

EP 1315066 A1 5/2003
JP H08-44677 2/1996
JP 2009-500706 1/2009

OTHER PUBLICATIONS

Morris, et al., "Specification for related U.S. Appl. No. 13/372,433," filed Feb. 13, 2012, p. 58 to be published in: US.

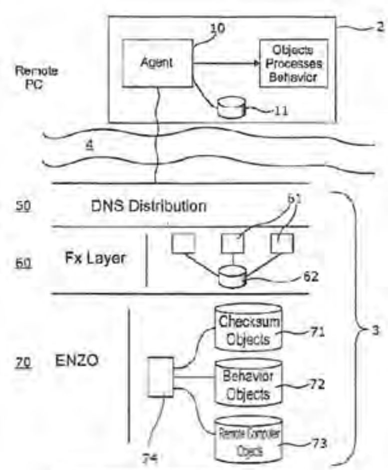
(Continued)

Primary Examiner — Dede Zecher
Assistant Examiner — Jason C Chiang
(74) *Attorney, Agent, or Firm* — Merchant & Gould P.C.

(57) **ABSTRACT**

Methods for classifying computer objects as malware and the associated apparatus are disclosed. An exemplary method includes, at a base computer, receiving data about a computer object from each of plural remote computers on which the object or similar objects are stored or processed wherein the base computer comprises plural threat servers arranged to receive the data from the plural remote computers and apply rules or heuristics against the data in real time to determine whether or not the object is malware and to communicate the determination to the remote computers. The base computer includes at least one central server in communication with the threat servers and arranged to receive the data about objects from the threat servers to maintain a master database of data received about objects from all threat servers.

15 Claims, 15 Drawing Sheets



Copy provided by USPTO from the PIRS Image Database on 10-05-2021

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,921,063	B1 *	4/2011	Quinlan	706/12
7,921,459	B2	4/2011	Houston et al.	
8,042,184	B1	10/2011	Batenin	
8,116,732	B2 *	2/2012	Issa et al.	455/410
8,146,135	B2	3/2012	Waissbein et al.	
8,195,938	B2 *	6/2012	Fanton et al.	713/165
8,225,218	B2	7/2012	Frayman et al.	
8,234,709	B2 *	7/2012	Viljoen et al.	726/24
8,276,202	B1 *	9/2012	Dubrovsky et al.	726/22
8,296,178	B2 *	10/2012	Hudis et al.	370/401
8,301,904	B1 *	10/2012	Gryaznov	713/188
8,307,435	B1	11/2012	Mann et al.	
8,321,540	B2 *	11/2012	Webb-Johnson	709/221
8,321,944	B1 *	11/2012	Mayer et al.	726/25
8,332,946	B1 *	12/2012	Boisjolie et al.	726/24
8,336,100	B1 *	12/2012	Glick et al.	726/23
8,347,375	B2 *	1/2013	Graham et al.	726/13
8,347,386	B2 *	1/2013	Mahaffey et al.	726/23
8,352,484	B1 *	1/2013	Schneider	707/758
8,407,790	B2	3/2013	Mony	
8,443,447	B1 *	5/2013	Chen et al.	726/24
8,448,243	B1 *	5/2013	Sankruthi	726/22
8,453,244	B2 *	5/2013	Dai et al.	726/24
8,499,283	B2	7/2013	Mony	
8,543,694	B2	9/2013	Petersen et al.	
2003/0093429	A1 *	5/2003	Nishikawa et al.	707/10
2003/0188194	A1	10/2003	Currie et al.	
2004/0103315	A1	5/2004	Cooper et al.	
2005/0131959	A1	6/2005	Thorman et al.	
2005/0172338	A1	8/2005	Sandu et al.	
2005/0187740	A1	8/2005	Marinescu	
2006/0069912	A1	3/2006	Zheng et al.	
2006/0080637	A1	4/2006	Treit et al.	
2006/0090073	A1	4/2006	Steinberg et al.	
2006/0101334	A1	5/2006	Liao et al.	
2006/0117055	A1	6/2006	Doyle	
2006/0161856	A1	7/2006	Heir	
2006/0272020	A1	11/2006	Gardner	
2007/0016953	A1	1/2007	Morris et al.	
2007/0028304	A1 *	2/2007	Brennan	726/24
2007/0174429	A1	7/2007	Mazzaferri et al.	
2007/0220043	A1	9/2007	Oliver et al.	
2008/0109473	A1	5/2008	Dixon et al.	
2008/0115190	A1	5/2008	Aaron	
2008/0148381	A1	6/2008	Aaron	
2008/0229422	A1	9/2008	Hudis et al.	
2009/0070304	A1 *	3/2009	Chen et al.	707/3
2009/0089040	A1	4/2009	Monastyrsky et al.	
2009/0172774	A1	7/2009	Koti et al.	
2009/0292791	A1	11/2009	Livshits et al.	
2009/0300764	A1	12/2009	Freeman	
2010/0043072	A1	2/2010	Rothwell	
2010/0064368	A1	3/2010	Stolfo et al.	
2010/0306851	A1	12/2010	Zhou	
2010/0313183	A1	12/2010	Ellen et al.	
2010/0332593	A1 *	12/2010	Barash et al.	709/203
2011/0047594	A1	2/2011	Mahaffey et al.	
2011/0083183	A1	4/2011	Freeman	
2011/0138361	A1	6/2011	McEntee et al.	
2011/0191299	A1 *	8/2011	Huynh Huu et al.	707/646
2011/0191341	A1 *	8/2011	Meyer et al.	707/736
2011/0197177	A1	8/2011	Mony	
2011/0197272	A1	8/2011	Mony	
2012/0017200	A1	1/2012	Ghosh et al.	
2012/0260304	A1	10/2012	Morris et al.	
2012/0266208	A1	10/2012	Morris et al.	

OTHER PUBLICATIONS

Morris, et al., "Specification for related U.S. Appl. No. 13/372,439," filed Feb. 13, 2012, p. 54 to be published in: US.

Sogno-Pabis, Elzbieta, "Partial International Search Report re application PCT/EP2012/052497", May 4, 2012, p. 7, Published in: NL.

Clarke, "Fuzzing for Software Vulnerability Discovery—Technical Report RHUL-MA-2009-04," Department of Mathematics, Royal Holloway, University of London, Feb. 17, 2009, 178 pages. Retrieved from: www.ma.rhul.ac.uk/static/techrep/2009/RHUL-MA-2009-04.pdf.

Su et al., "The Essence of Command Injection Attacks in Web Applications," Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '06), 2006, pp. 372-382.

Official Action for U.S. Appl. No. 12/703,074 mailed Sep. 12, 2012, 16 pages.

Notice of Allowance for U.S. Appl. No. 12/703,074 mailed Nov. 21, 2012, 5 pages.

Notice of Allowance for U.S. Appl. No. 13/010,639 mailed Mar. 20, 2013, 15 pages.

Official Action for U.S. Appl. No. 13/372,433 mailed Apr. 24, 2013, 6 pages.

Official Action for U.S. Appl. No. 13/372,439 mailed Apr. 22, 2013, 6 pages.

Extended European Search Report for EP Application No. 14161844.7, mailed Jul. 25, 2014, 5 pages.

Extended European Search Report for EP Application No. 14161847.0, mailed Jul. 25, 2014, 5 pages.

International Search Report and Written Opinion for International (PCT) Patent Application No. PCT/EP2012/052497 mailed Jul. 13, 2012, 22 pages.

International Preliminary Report on Patentability for International (PCT) Patent Application No. PCT/EP2012/052497 mailed Aug. 29, 2013, 16 pages.

"Network Security Task Manager: Manual—Installation—Configuration," A. & M. Neuber Software GmbH, 2008, 65 pages.

Chacksfield, "Facebook apps ask for your permission before data mining," TechRadar.com, Jul. 1, 2010 [retrieved on Jun. 14, 2013], 5 pages. Retrieved from: www.techradar.com/us/news/broadband/web/internet/facebook-apps-ask-for-your-permission-before-data-mining-700540.

Cogswell et al., "RootkitRevealer v1.71," Nov. 1, 2006, [Retrieved on Jun. 29, 2012], 5 pages. Retrieved from: http://technet.microsoft.com/en-us/sysinternals/bb897445.aspx.

Srinivasan, "Protecting Anti-Virus Software Under Viral Attacks," Master of Science thesis, Arizona State University, Aug. 2007, 82 pages.

Official Action for U.S. Appl. No. 13/372,433 mailed Aug. 9, 2013, 18 pages.

Official Action for U.S. Appl. No. 13/372,439 mailed Jul. 3, 2013, 16 pages.

Dietrich (2003) Code Project "Event Logging, Part I: XEventLog—Basic NT Event Logging," 10 pages. Available at: http://www.codeproject.com/Articles/3995/Event-Logging-Part-I-XEventLog-Basic-NT-Event-Logg.

Wikipedia as crawled by Wayback Machine on Mar. 1, 2010 "Event Viewer" 6 pages.

Chinese Patent Application No. 201280018306.9, First Office Action mailed Jun. 2, 2015, 26 pages. (No English translation available).

U.S. Appl. No. 12/703,074, Amendment and Response dated Nov. 6, 2012, 8 pages.

U.S. Appl. No. 12/703,074, Amendment after Allowance dated Dec. 7, 2012, 7 pages.

U.S. Appl. No. 13/372,433, Amendment and Response dated Jan. 6, 2014, 13 pages.

U.S. Appl. No. 13/372,433, Final Rejection mailed May 9, 2014, 22 pages.

U.S. Appl. No. 13/372,433, Amendment and Response dated Oct. 9, 2014, 14 pages.

U.S. Appl. No. 13/372,433, Non-Final Rejection mailed Sep. 10, 2015, 16 pages.

U.S. Appl. No. 13/372,439, Amendment and Response dated Nov. 4, 2013, 12 pages.

U.S. Appl. No. 13/372,439, Final Rejection mailed Feb. 24, 2014, 15 pages.

U.S. Appl. No. 13/372,433, Amendment and Response filed Feb. 10, 2016, 18 pages.

Japanese Patent Application No. 2013-553909, Office Action mailed Jan. 26, 2016, 6 pages (English Translation).

Chinese Patent Application No. 201280018306.9, Second Office Action mailed Jan. 11, 2016, 11 pages (English Translation).

Australian Patent Application No. 2012217181, First Examination Report mailed Apr. 11, 2016, 4 pages.

Japanese Patent Application No. 2013-553909, Notice of Allowance mailed May 24, 2016, 3 pages (No English translation available).

U.S. Appl. No. 13/372,433, Final Rejection mailed Jun. 1, 2016, 17 pages.

* cited by examiner

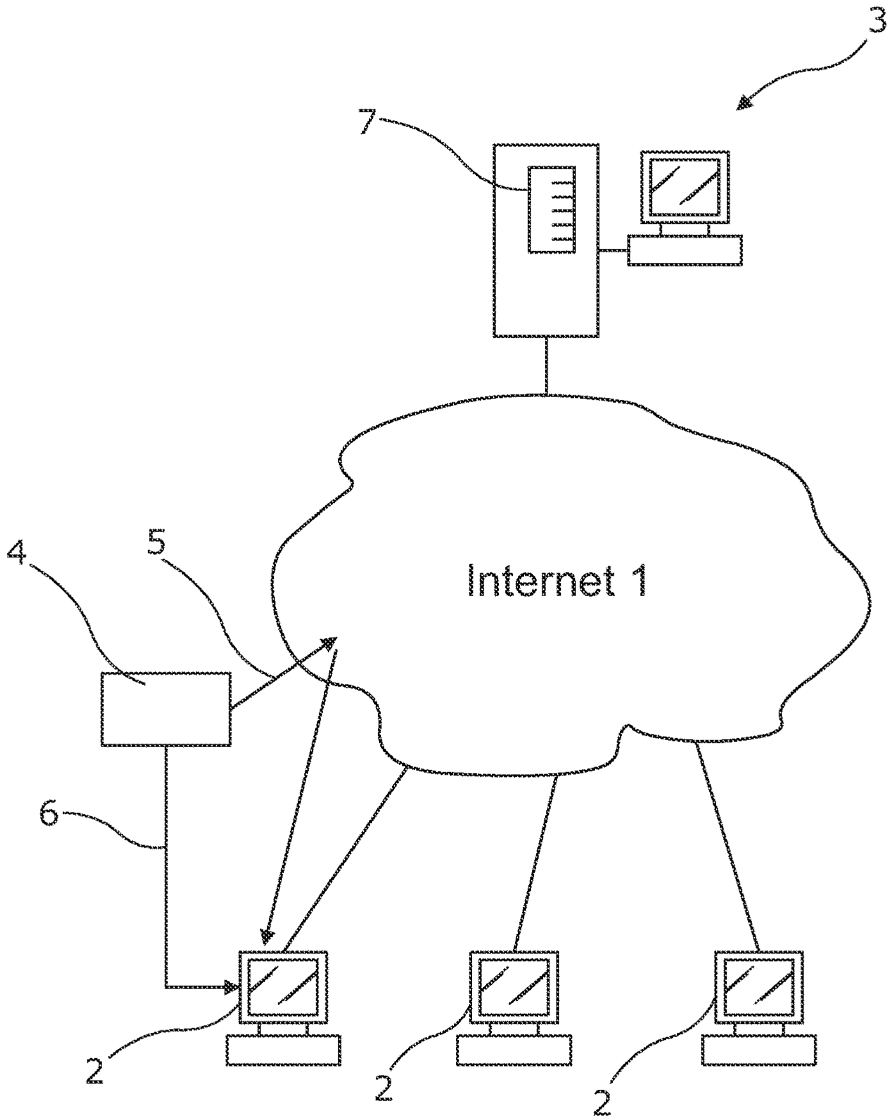


Fig. 1
(Prior Art)

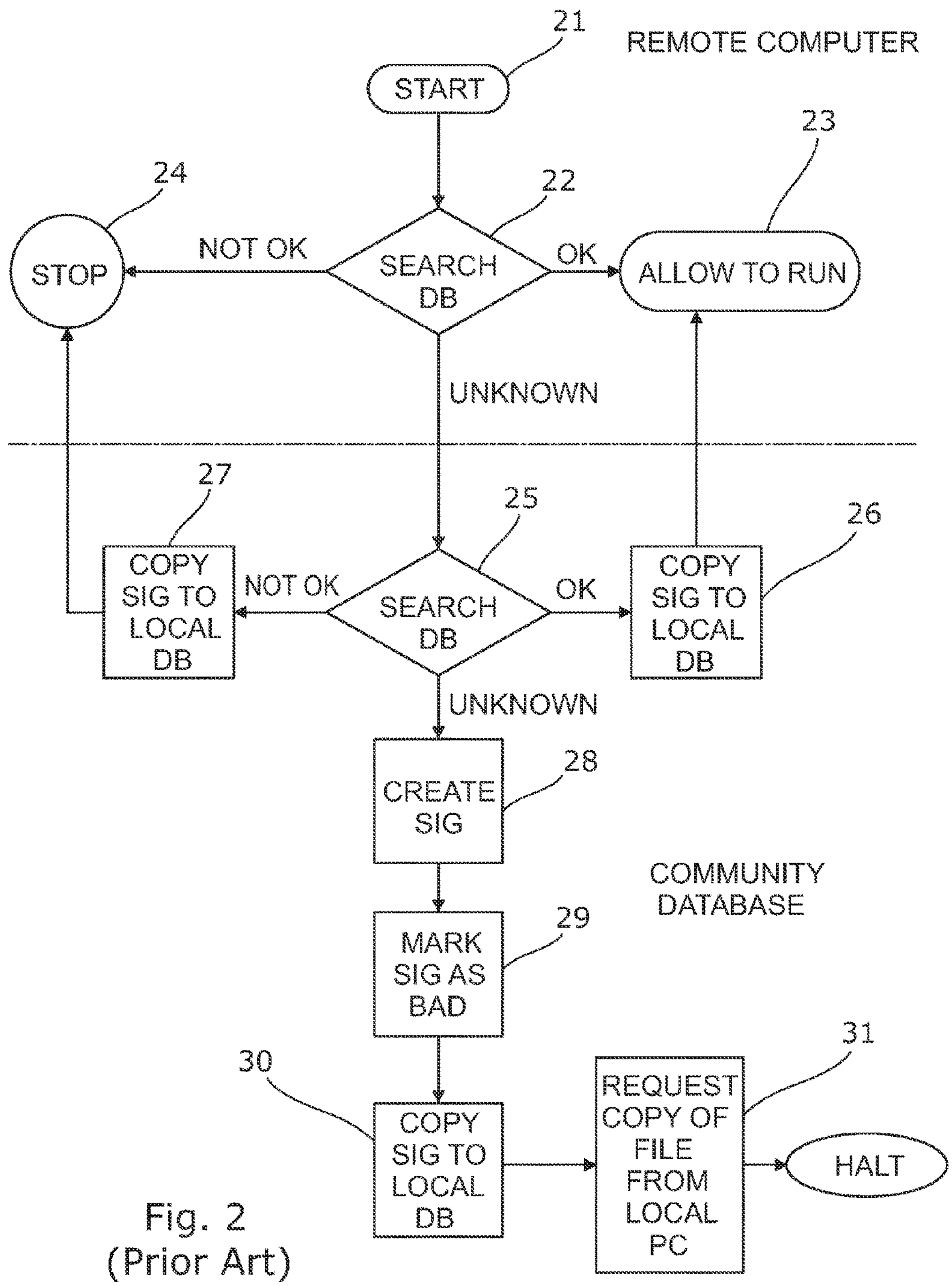
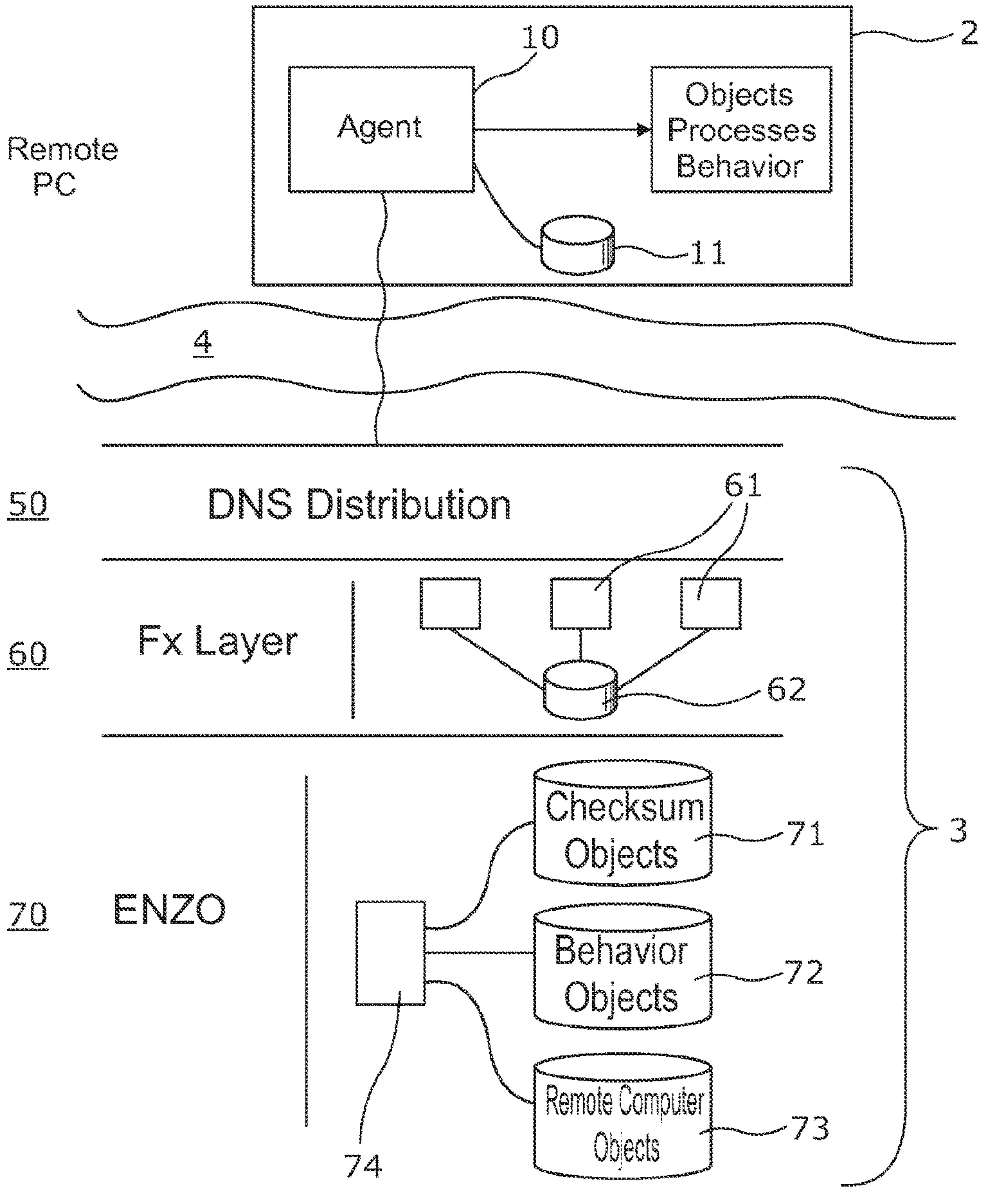


Fig. 2
(Prior Art)



Remote PC
10
2
Agent
Objects Processes Behavior
11
4
50
DNS Distribution
61
60
Fx Layer
62
70
ENZO
Checksum Objects
71
Behavior Objects
72
Remote Computer Objects
73
74
3

Fig. 3

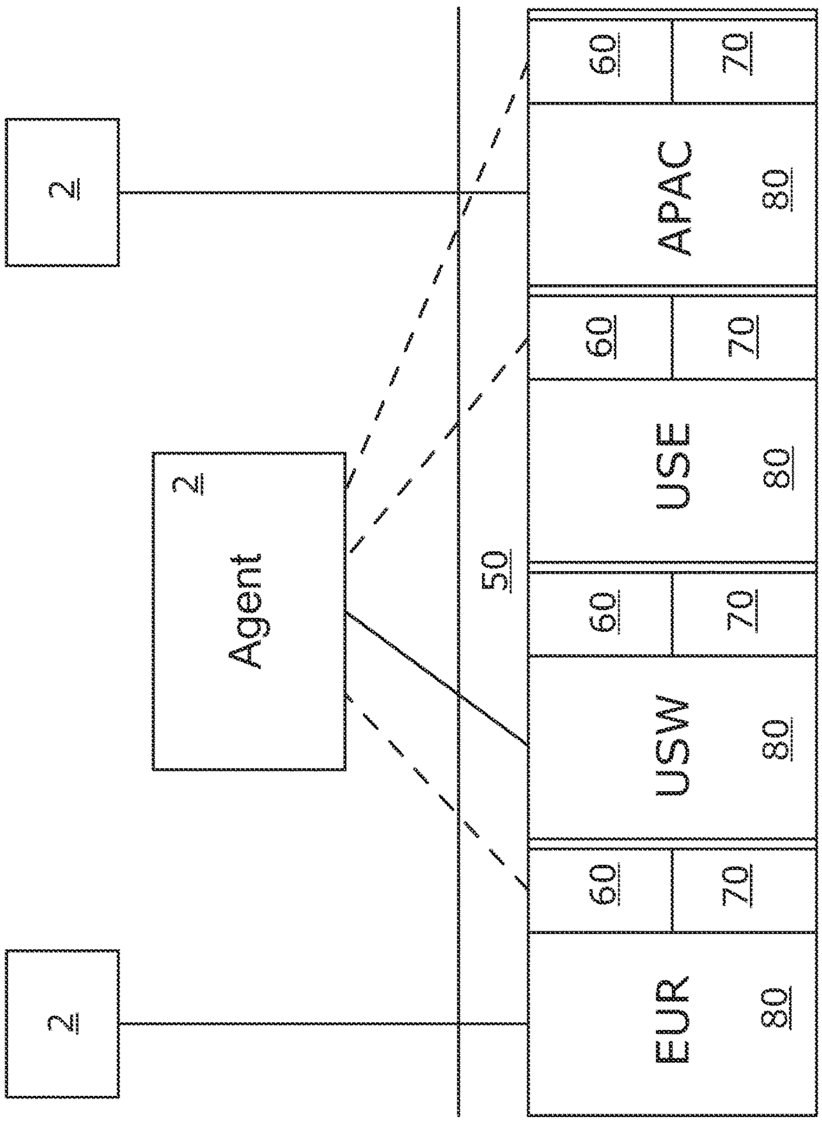
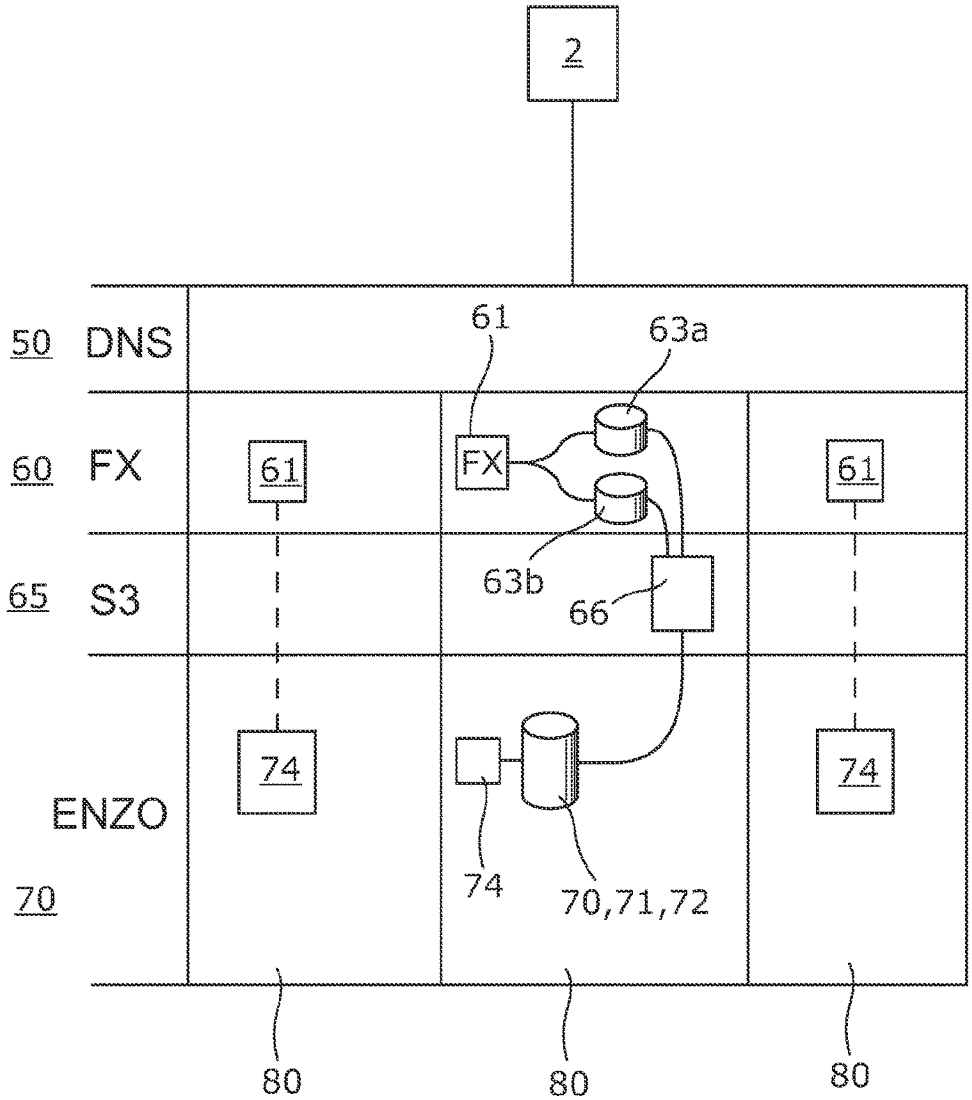


Fig. 4



2
61
63a
50 DNS
60 FX
61
FX
61
63b
66
65 S3
74
74
74
ENZO
70
74
70,71,72
80
80
80

Fig. 5

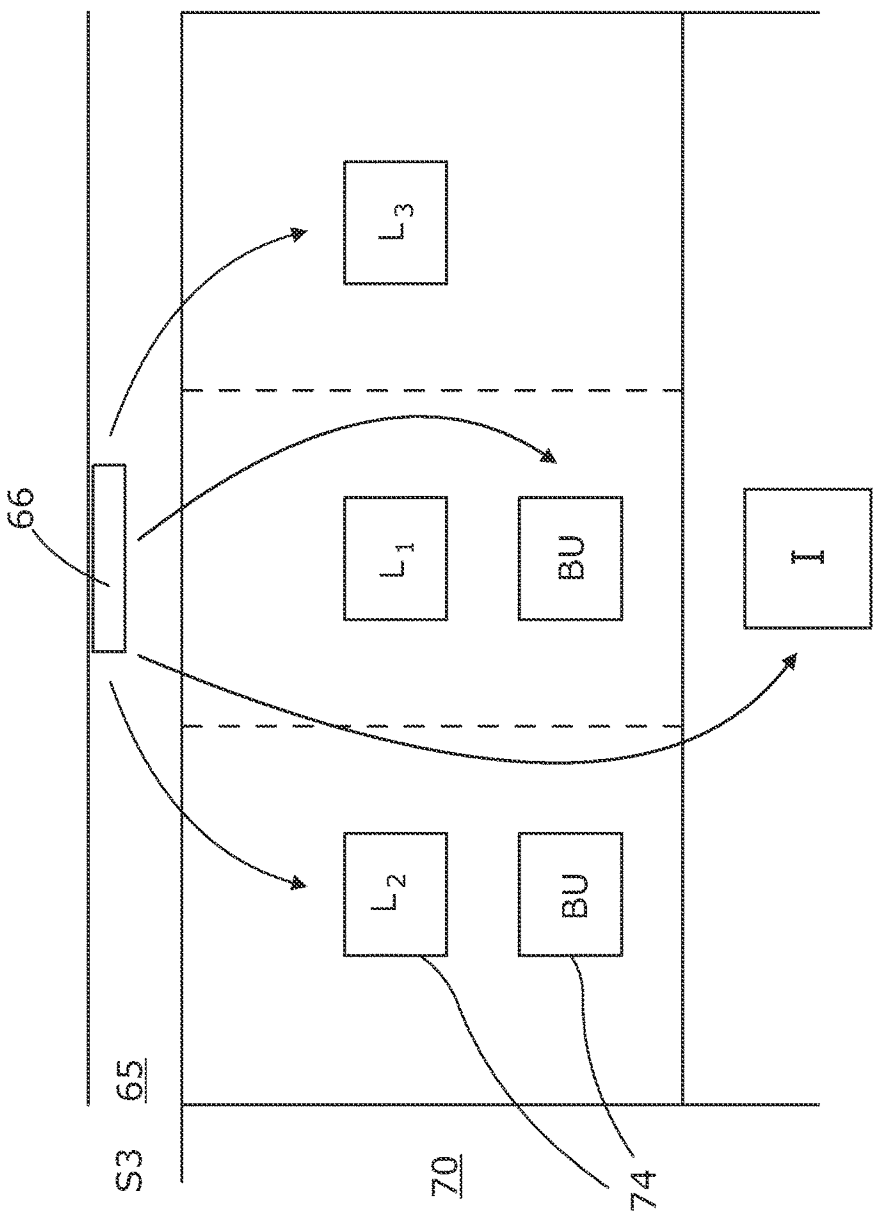


Fig. 6

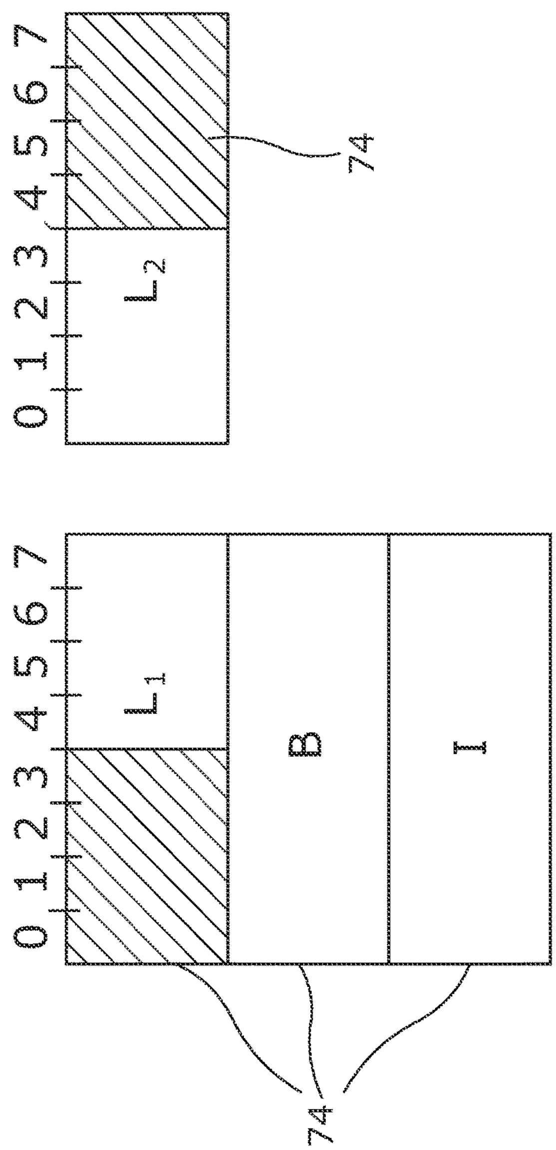


Fig. 7

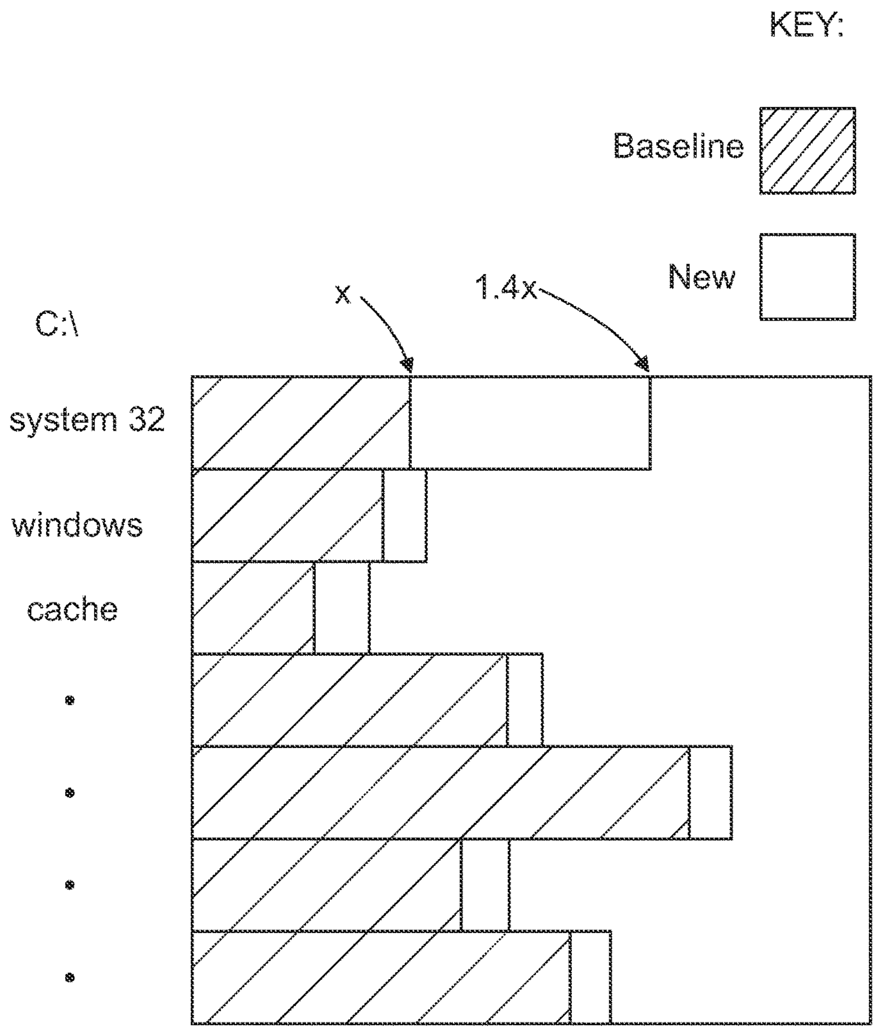


Fig. 8

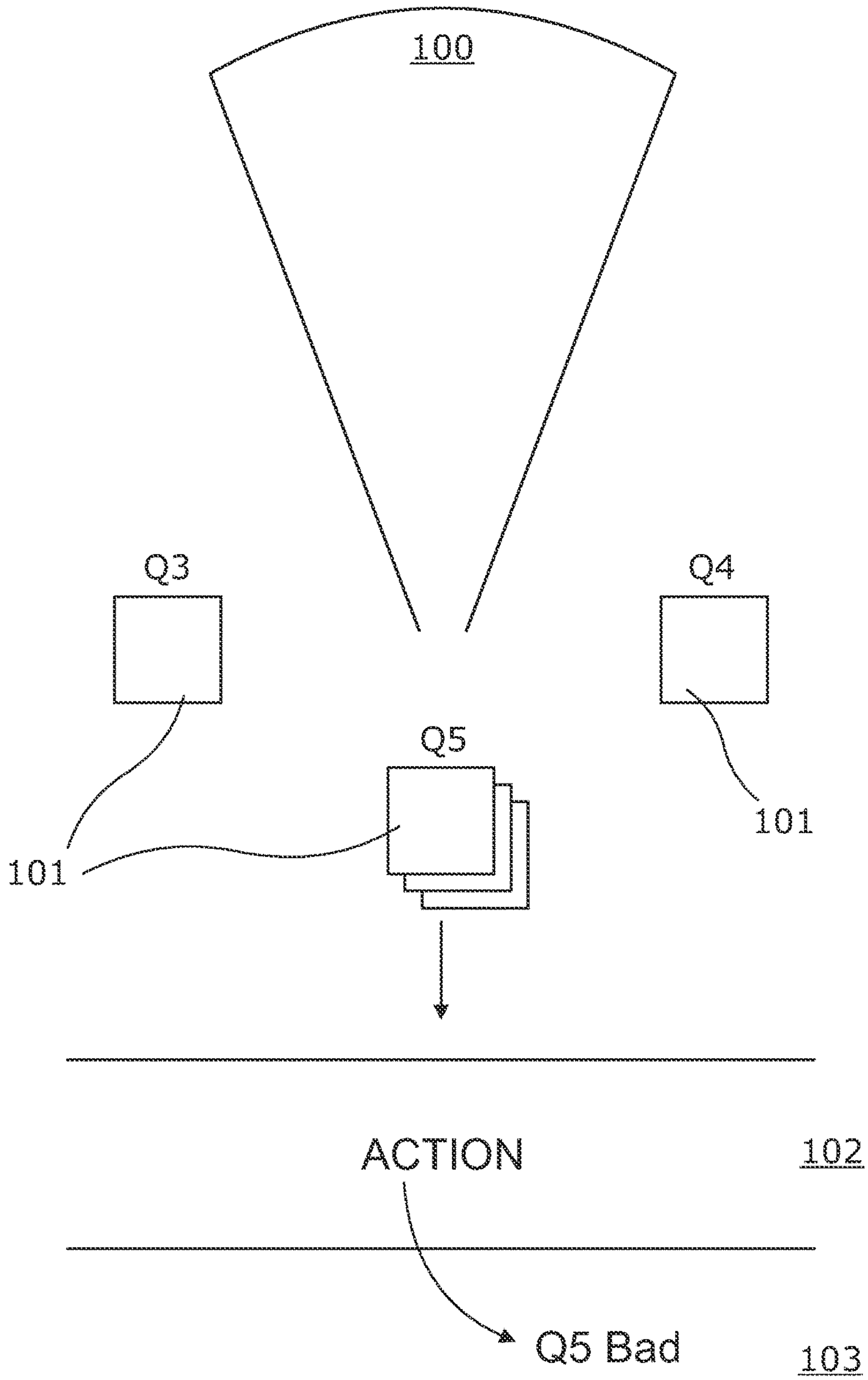


Fig. 9

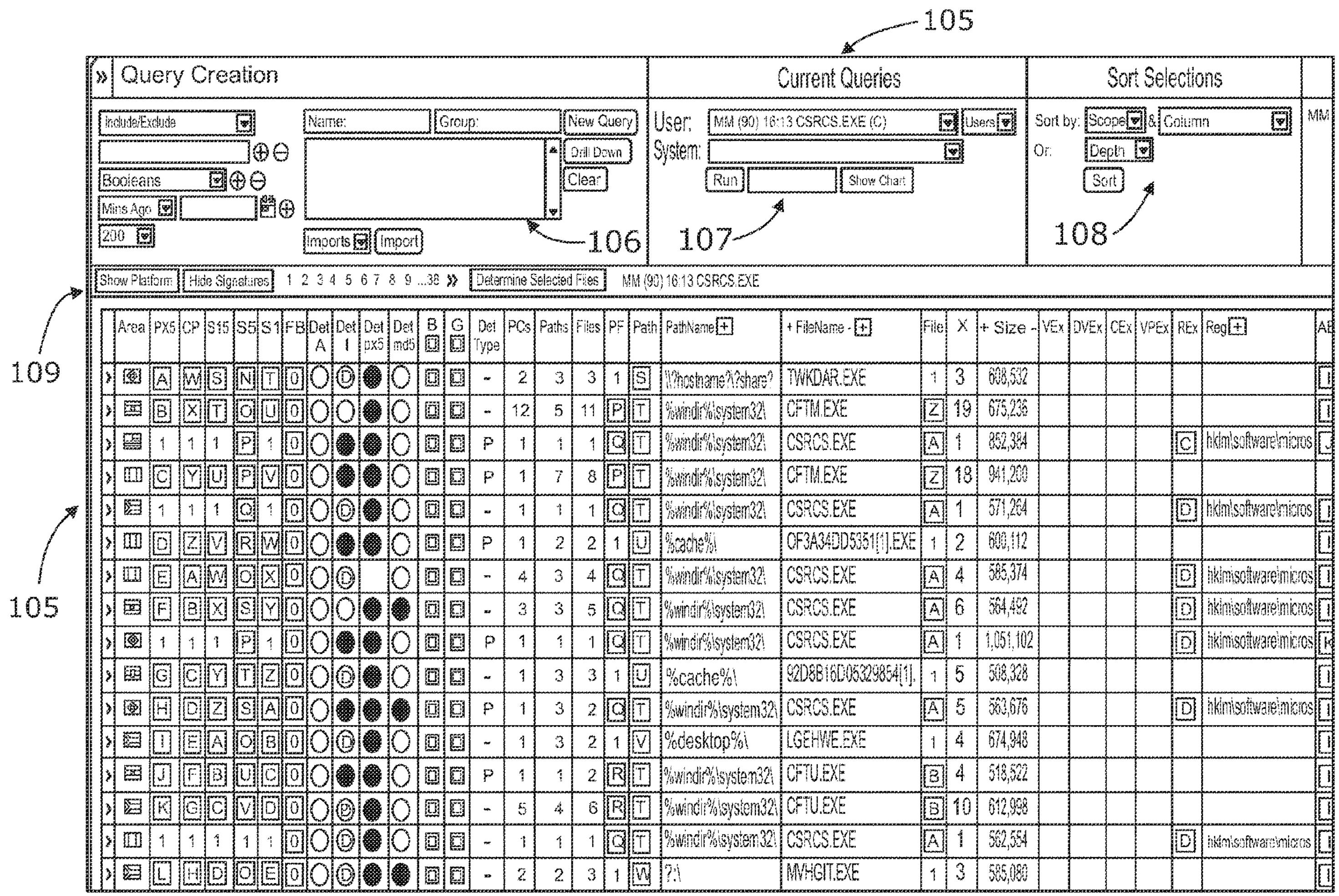


Fig. 10

WBRT007861

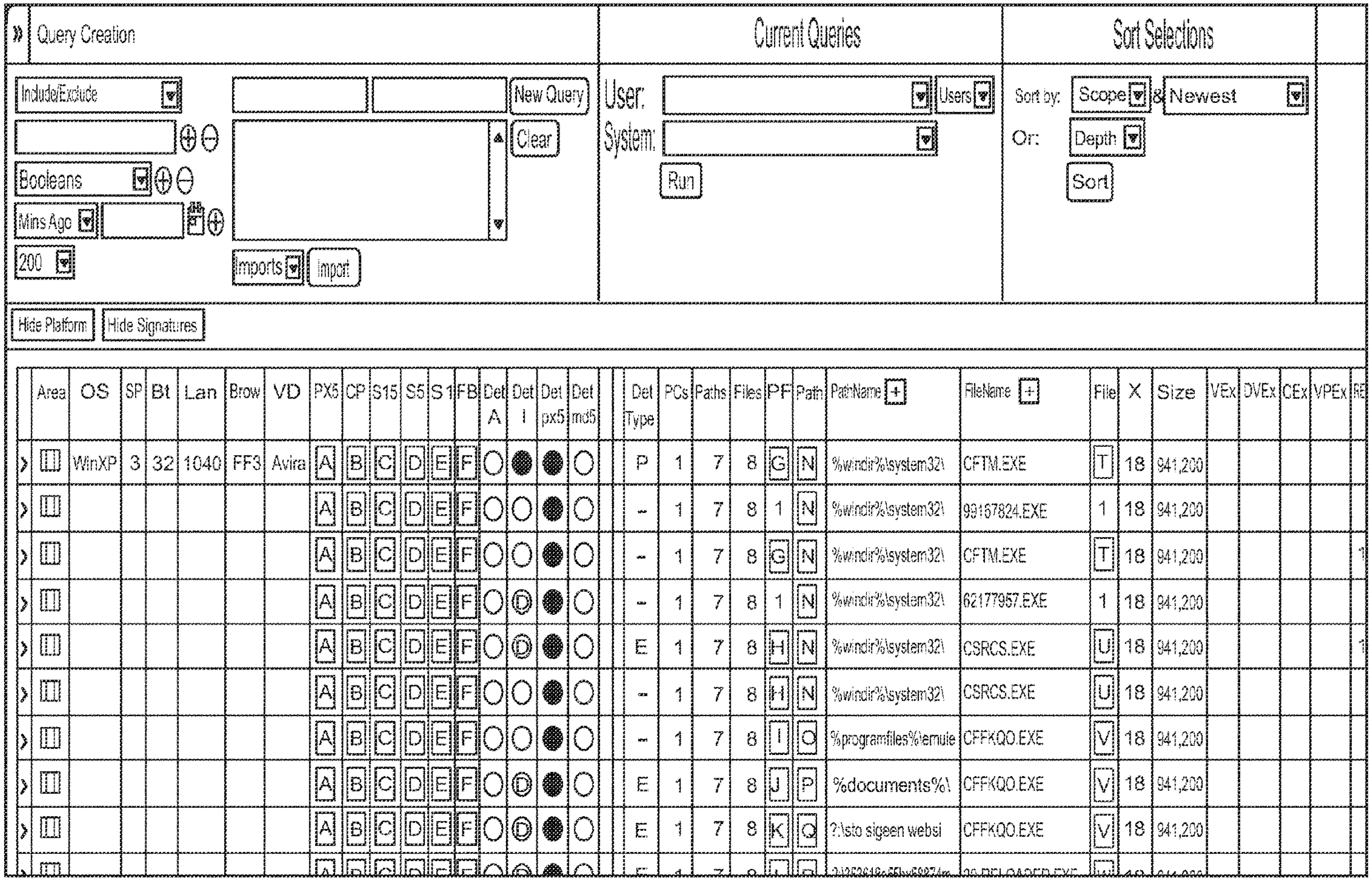

Query Creation
Current Queries
Sort Selections
Include/Exclude
New Query
Clear
Booleans
Mins Ago
200
Imports
Import
User:
Users
System:
Run
Sort by:
Scope
Newest
Or:
Depth
Sort
Hide Platform
Hide Signatures
Area OS SP Bt Lan Brow VD PX5 CP S15 S5 S1 FB Det A Det I Det px5 Det md5 Det Type PCs Paths Files PF Path PathName FileName File X Size VEx DVEx CEx VPEx
WinXP 3 32 1040 FF3 Avira A B C D E F P 1 7 8 G N %windir%\system32\ CFTM.EXE T 18 941,200
A B C D E F - 1 7 8 1 N %windir%\system32\ 99167824.EXE 1 18 941,200
A B C D E F - 1 7 8 G N %windir%\system32\ CFTM.EXE T 18 941,200
A B C D E F - 1 7 8 1 N %windir%\system32\ 62177957.EXE 1 18 941,200
A B C D E F E 1 7 8 H N %windir%\system32\ CSRCS.EXE U 18 941,200
A B C D E F - 1 7 8 H N %windir%\system32\ CSRCS.EXE U 18 941,200
A B C D E F - 1 7 8 I O %programfiles%\emule CFFKQO.EXE V 18 941,200
A B C D E F E 1 7 8 J P %documents%\ CFFKQO.EXE V 18 941,200
A B C D E F E 1 7 8 K Q ?:\sto sigeen websi CFFKQO.EXE V 18 941,200

Fig. 11

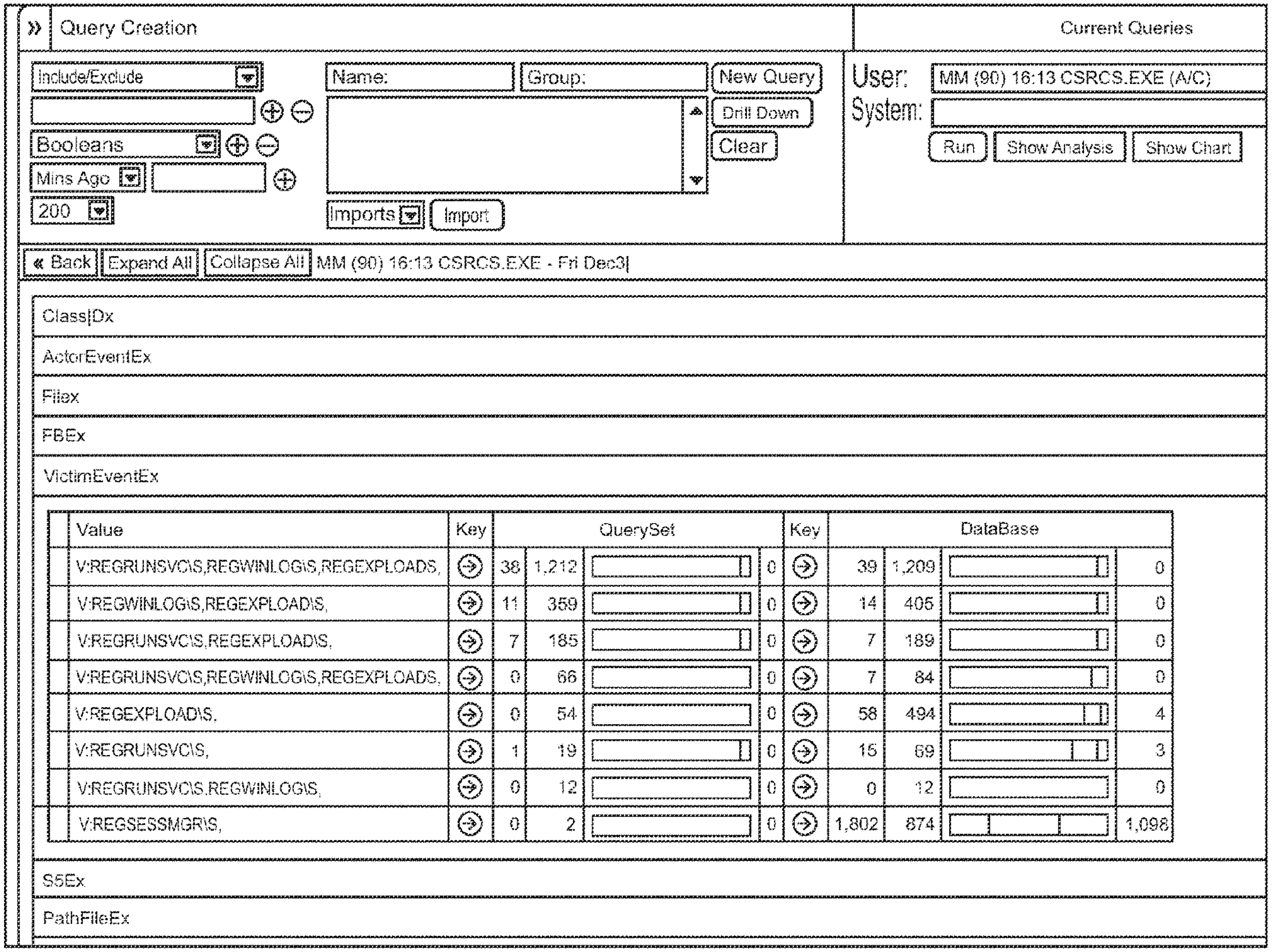

Query Creation
Current Queries
Include/Exclude
Name:
Group:
New Query
User:
MM (90) 16:13 CSRCS.EXE (A/C)
System:
Drill Down
Booleans
Clear
Run
Show Analysis
Show Chart
Mins Ago
200
Imports
Import
« Back
Expand All
Collapse All
MM (90) 16:13 CSRCS.EXE - Fri Dec3|
ClassIDx
ActorEventEx
Filex
FBEx
VictimEventEx
Value
Key
QuerySet
Key
DataBase
V:REGRUNSVC\S,REGWINLOG\S,REGEXPLOADS, 38 1,212 0 39 1,209 0
V:REGWINLOG\S,REGEXPLOAD\S, 11 359 0 14 405 0
V:REGRUNSVC\S,REGEXPLOAD\S, 7 185 0 7 189 0
V:REGRUNSVC\S,REGWINLOG\S,REGEXPLOADS, 0 66 0 7 84 0
V:REGEXPLOAD\S, 0 54 0 58 494 4
V:REGRUNSVC\S, 1 19 0 15 69 3
V:REGRUNSVC\S,REGWINLOG\S, 0 12 0 0 12 0
V:REGSESSMGR\S, 0 2 0 1,802 874 1,098
S5Ex
PathFileEx

Fig. 12

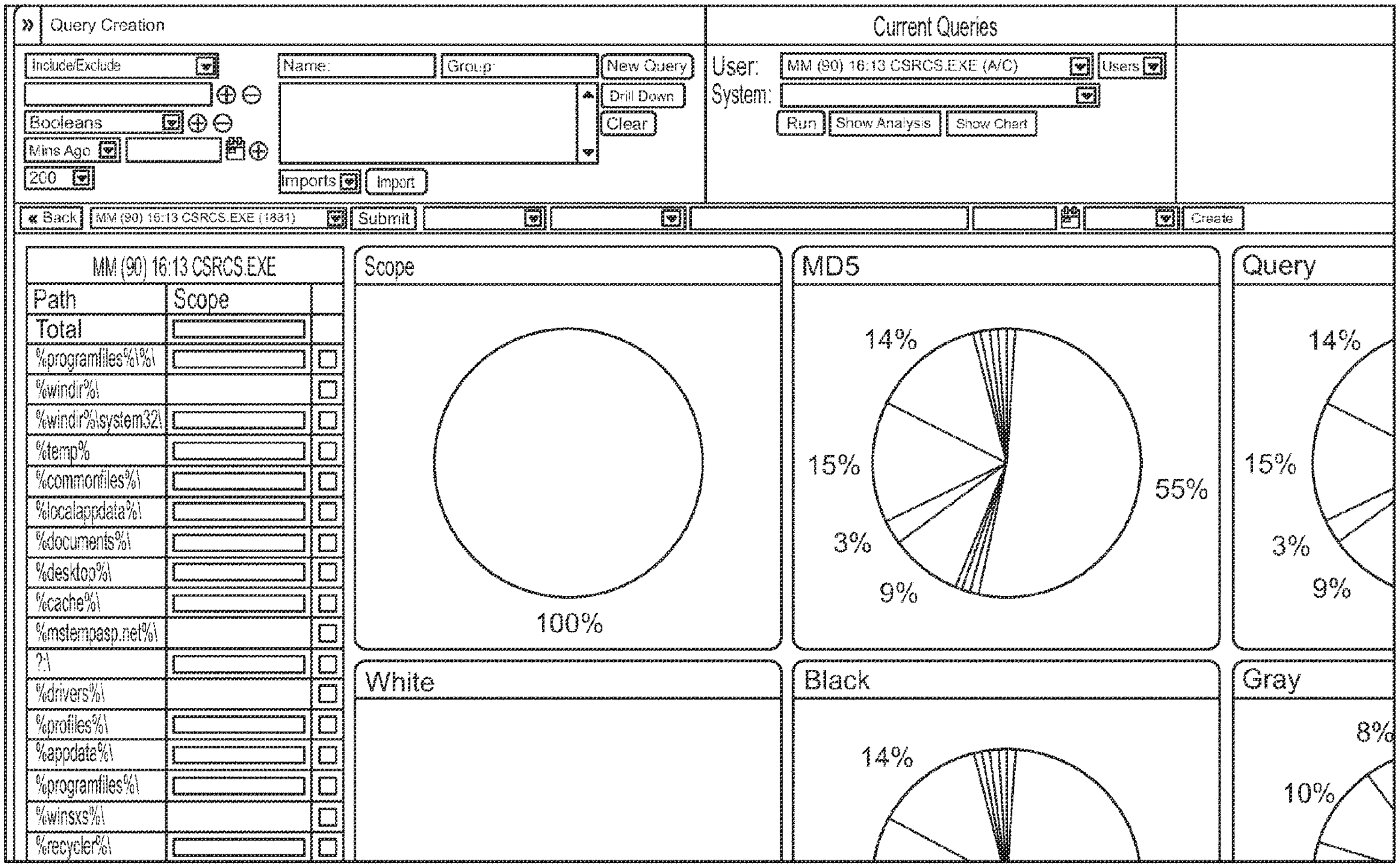

Query Creation
Current Queries
Include/Exclude
Booleans
Mins Ago
200
Name:
Group:
New Query
Drill Down
Clear
Imports
Import
User:
System:
MM (90) 16:13 CSRCS.EXE (A/C)
Users
Run
Show Analysis
Show Chart
« Back
MM (90) 16:13 CSRCS.EXE (1831)
Submit
Create
MM (90) 16:13 CSRCS.EXE
Path
Scope
Total
%programfiles%\%\
%windir%\
%windir%\system32\
%temp%
%commonfiles%\
%localappdata%\
%documents%\
%desktop%\
%cache%\
%mstempasp.net%\
?:\
%drivers%\
%profiles%\
%appdata%\
%programfiles%\
%winsxs%\
%recycler%\
Scope
100%
MD5
14%
15%
3%
9%
55%
Query
14%
15%
3%
9%
White
Black
14%
Gray
8%
10%

Fig. 13

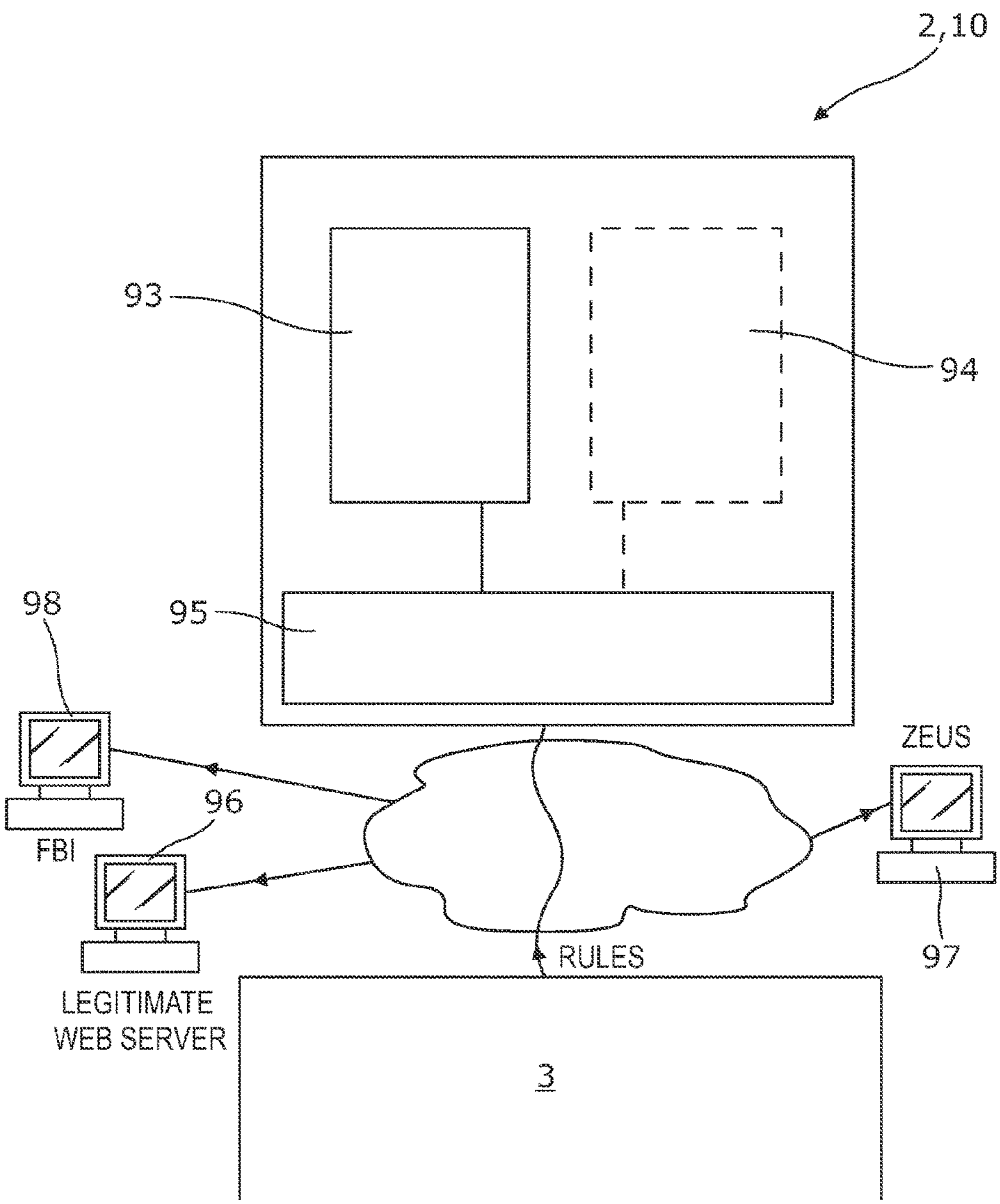


Fig. 14

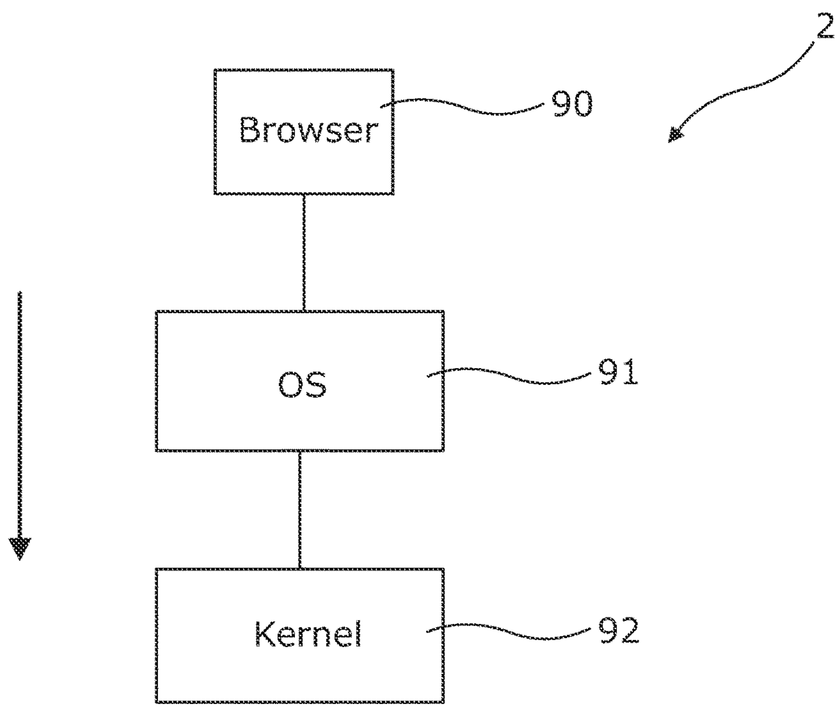


Fig. 15

METHODS AND APPARATUS FOR DEALING WITH MALWARE

CLAIM OF PRIORITY UNDER 35 U.S.C. §119

The present Application for Patent claims priority to Provisional Application No. 61/443,095 entitled "METHODS AND APPARATUS FOR DEALING WITH MALWARE" filed Feb. 15, 2011, and assigned to the assignee hereof and hereby expressly incorporated by reference herein.

REFERENCE TO CO-PENDING APPLICATIONS FOR PATENT

The present Application for Patent is related to the following U.S. Patent Applications: co-pending U.S. patent application Ser. No. 13/372,433, entitled "METHODS AND APPARATUS FOR MALWARE THREAT RESEARCH," assigned to the assignee hereof, and expressly incorporated by reference herein; and U.S. patent application Ser. No. 13/372,439, entitled "METHODS AND APPARATUS FOR AGENT-BASED MALWARE MANAGEMENT", now abandoned, assigned to the assignee hereof, and expressly incorporated by reference herein.

BACKGROUND

1. Field

The present invention relates generally to methods and apparatus for dealing with malware. And more specifically, systems and methods for protection against malware.

2. Background

The term "malware" is used herein to refer generally to any executable computer file or, more generally "object", that is or contains malicious code, and thus includes viruses, Trojans, worms, spyware, adware, etc. and the like.

A typical anti-malware product, such as virus scanning software, scans objects or the results of an algorithm applied to the object or part thereof to look for signatures in the object that are known to be indicative of the presence of malware. Generally, the method of dealing with malware is that when new types of malware are released, for example via the Internet, these are eventually detected. Once new items of malware have been detected, then the service providers in the field generate signatures that attempt to deal with these and these signatures are then released as updates to their anti-malware programs. Heuristic methods have also been employed.

These systems work well for protecting against known malicious objects. However, since they rely on signature files being generated and/or updated, there is inevitably a delay between a new piece of malware coming into existence or being released and the signatures for identifying that malware being generated or updated and supplied to users. Thus, users are at risk from new malware for a certain period of time which might be up to a week or even more.

More recently, so-called "cloud" based techniques have been developed for fighting malware/viruses. In these techniques, protection is provided by signatures that are stored in the cloud, i.e. in a central server to which the remote computers are connected. Thus, a remote computer can be given protection as soon as a new malware object is spotted and its signature stored in the central server, so that the remote computer is protected from it without the need to wait for the latest signatures to be downloaded and installed on the remote computer. This technique can also give the advantage of moving the processing burden from the remote computer to the central server. However, this technique is limited by sending only signature information or very basic information about an object to the central server. Therefore, in order to analyse whether or not an unknown object is malware, a copy of that object is normally sent to the central server where it is investigated by a human analyst. This is a time consuming and laborious task introducing considerable delay in classifying malware as safe or unsafe. Also, given the considerable volume of new objects that can be seen daily across a community, it is unrealistic to have a skilled human analyst investigate each new object thoroughly. Accordingly, malevolent objects may escape investigation and detection for considerable periods of time during which time they can carry out their malevolent activity in the community.

We refer in the following to our previous application US-A-2007/0016953, published 18 Jan. 2007, entitled "METHODS AND APPARATUS FOR DEALING WITH MALWARE," the entire contents of which are hereby incorporated by reference. In this document, various new and advantageous cloud-based strategies for fighting malware are disclosed. In particular a cloud-based approach is outlined where the central server receives information about objects and their behaviour on remote computers throughout the community and builds a picture of objects and their behaviour seen throughout the community. This information is used to make comparisons with this data across the community in developing and applying various heuristics and or rules to determine whether a newly seen object is malevolent or not.

This approach to fighting malware involves communicating, storing and managing vast amounts of data at the central server, which is a challenging problem in itself. It is also challenging to develop new schemes for more accurately and more efficiently detecting malware given the vast amount of data collected about objects seen in the community and the constantly evolving strategies used by malware writers to evade detection. It is also desirable to improve the processes for analysing the data to make the best use of the time and specialised skills of the human malware analysts.

Malware is also becoming increasingly adept at self-defence by interfering with the operation of security programs installed on a computer. This is another problem that security software must contend with.

SUMMARY

According to a first aspect of the present invention, there is provided a method of classifying a computer object as malware, the method includes:

at a base computer, receiving data about a computer object from each of plural remote computers on which the object or similar objects are stored and or processed;

wherein the base computer comprises plural threat servers arranged to receive the data from the plural remote computers and apply rules and or heuristics against that data in real time to determine whether or not the object is malware and to communicate the determination to the remote computers;

the base computer further comprising at least one central server in communication with the threat servers and arranged to receive the data about objects from the threat servers to maintain a master database of data received about objects from all of the threat servers.

According to another aspect, an apparatus is provided for classifying a computer object as malware, the apparatus includes a base computer arranged to receive data about a computer object from each of plural remote computers on which the object or similar objects are stored or processed.

The base computer includes plural threat servers arranged to receive the data from the plural remote computers and apply rules or heuristics against that data in real time to determine whether or not the object is malware and to communicate the determination to the remote computers. The base computer also includes at least one central server in communication with the threat servers and arranged to receive the data about objects from the servers to maintain a master database of data received about objects from all threat servers.

According to yet another aspect, a method is provided for connecting a remote computer to one of a plurality of servers. The method includes installing an agent program on the remote computer, the agent program having an algorithm for allocating the remote computer to a server determined by the time when the agent program was installed; and connecting the agent program to the allocated server.

As will become apparent in view of the following disclosure, the various aspects and embodiments of the invention can be combined.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows schematically apparatus in which an embodiment of the present invention may be implemented;

FIG. **2** is a flowchart showing schematically the operation of an example of a method according to an embodiment of the present invention;

FIG. **3** shows a more detailed view of an example of a base computer **3** according to an embodiment of the present invention;

FIG. **4** shows an example where web servers are located in different geographical locations;

FIG. **5** shows schematically the apparatus of FIG. **3** in more detail and in particular the way in which data can be moved from the FX layer to the ENZO layer in an embodiment;

FIG. **6** shows schematically an example in which a new instance of an ENZO server is created in accordance with an embodiment of the invention;

FIG. **7** shows schematically an example of apparatus by which an ENZO server shares out its workload between plural servers in accordance with an embodiment of the invention;

FIG. **8** is a chart showing an example of a scheme for analysing data about objects according to an embodiment of the present invention;

FIG. **9** show shows schematically an example of a scheme for processing data about objects according to an embodiment of the invention;

FIGS. **10** to **13** show graphical user interfaces of an example of a computer program for researching objects according to an embodiment of the present invention;

FIG. **14** shows an example of an agent program on a remote computer according to an embodiment of the present invention; and

FIG. **15** shows the hierarchy on a computer system.

DETAILED DESCRIPTION

Overview

Referring to FIG. **1**, a computer network is generally shown as being based around a distributed network such as the Internet **1**. Embodiments of the present invention may however be implemented across or use other types of network, such as a LAN. Plural local or "remote" computers **2** are connected via the Internet **1** to a "central" or "base" computer **3**. The computers **2** may each be variously a personal computer, a server of any type, a PDA, mobile phone, an interactive television, or any other device capable of loading and operating computer objects. An object in this sense may be a computer file, part of a file or a sub-program, macro, web page or any other piece of code to be operated by or on the computer, or any other event whether executed, emulated, simulated or interpreted. An object **4** is shown schematically in the figure and may for example be downloaded to a remote computer **2** via the Internet **1** as shown by lines **5** or applied directly as shown by line **6**. The object **4** may reside in computer RAM, on the hard disk drive of the computer, on removable storage connected to the computer, such as a USB pen drive, an email attachment, etc.

In one exemplary embodiment, the base computer **3** is in communication with a database **7** with which the remote computers **2** can interact when the remote computers **2** run an object **4** to determine whether the object **4** is safe or unsafe. The community database **7** is populated, over time, with information relating to each object run on all of the connected remote computers **2**. As will be discussed further below, data representative of each object **4** may take the form of a so-called signature or key relating to the object, its attributes and behaviour.

Referring now to FIG. **2**, at the start point **21**, a computer object **4** such as a process is run at a remote computer **2**. At step **22**, by operation of a local "agent" program or software running on the remote computer **2**, the operation of the process is hooked so that the agent program can search a local database stored at the remote computer **2** to search for a signature or key representing that particular process, its related objects and/or the event. If the local signature is present, it will indicate either that the process is considered to be safe or will indicate that that process is considered unsafe. An unsafe process might be one that has been found to be malware or to have unforeseen or known unsafe or malevolent results arising from its running. If the signature indicates that the process is safe, then that process or event is allowed by the local agent program on the remote computer **2** to run at step **23**. If the signature indicates that the process is not safe, then the process or event is stopped at step **24**.

It will be understood that there may be more than two states than "safe" or "not-safe" and choices may be given to the user. For example, if an object is considered locally to be not safe, the user may be presented with an option to allow the related process to run nevertheless. It is also possible for different states to be presented to each remote computer **2**. The state can be varied by the base computer to take account of the location, status or ownership of the remote computer or timeframe.

Furthermore, the agent software at the remote computer **2** may be arranged to receive rules or heuristics from the base computer **3** for classifying objects as safe or unsafe. If the object is unknown locally, the agent software may apply the rules or heuristics to the details of the object to try to classify the object as safe or unsafe. If a classification is made, details of the object and classification are passed to the base computer **3** to be stored in the community database **7**. This means that the agent software is capable of providing protection against previously unseen objects even if it is "offline", e.g. it is unable to connect to the base computer for any reason. This mechanism is described in more detail later in the disclosure.

If the object is still not classified locally, then details of the object are passed over the Internet **1** or other network to the base computer **3** for storing in the community database **7** and for further analysis at the base computer **3**. In that case, the community database **7** is then searched at step **25** for a signature for that object that has already been stored in the community database **7**. The community database **7** is supplied with signatures representative of objects, such as pro-

grams or processes, run by each monitored remote computer **2**. In a typical implementation in the field, there may be several thousands or even millions of remote computers **2** connected or connectable to the base computer **3** and so any objects that are newly released upon the Internet **1** or that otherwise are found on any of these remote computers **2** will soon be found and signatures created and sent to the base computer **3** by the respective remote computers **2**.

When the community database **7** is searched for the signature of the object that was not previously known at the remote computer **2** concerned, then if the signature is found and indicates that that object is safe, then a copy of the signature or at least a message that the object is safe is sent to the local database of the remote computer **2** concerned at step **26** to populate the local database. In this way, the remote computer **2** has this information immediately to hand the next time the object **4** is encountered. A separate message is also passed back to the remote computer **2** to allow the object to run in the current instance.

If the signature is found in the community database **7** and this indicates for some reason that the object is unsafe, then again the signature is copied back to the local database and marked "unsafe" at step **27**, and/or a message is sent to the remote computer **2** so that running of the object is stopped (or it is not allowed to run) and/or the user given an informed choice whether to run it or not.

If after the entire community database **7** has been searched the object is still unknown, then it is assumed that this is an entirely new object which has never been seen before in the field. A signature is therefore created representative of the object at step **28**, or a signature sent by the remote computer **2** is used for this purpose.

At this point, rules or heuristics may be applied by the base computer **3** to the details of the object to try to classify the object as safe or unsafe. If a classification is made, the signature is marked as safe or unsafe accordingly in the community database **7**. The signature is copied to the local database of the remote computer **2** that first ran the object. This mechanism is described in more detail later in the disclosure.

If the object is still not classified, the signature may be initially marked as bad or unsafe in the community database **7** at step **29**. The signature is copied to the local database of the remote computer **2** that first ran the object at step **30**. A message may then be passed to the remote computer **2** to instruct the remote computer **2** not to run the object or alternatively the user may be given informed consent as to whether to allow the object to run or not. In addition, a copy of the object itself may be requested at step **31** by the community database **7** from the remote computer **2**.

If the user at the remote computer **2** chooses to run a process that is considered unsafe because it is too new, then that process may be monitored by the remote computer **2** and/or community database **7** and, if no ill effect occurs or is exhibited after a period of time of n days for example, it may then be considered to be safe. Alternatively, the community database **7** may keep a log of each instance of the process which is found by the many remote computers **2** forming part of the network and after a particular number of instances have been recorded, possibly with another particular number of instances or the process being allowed to run and running safely, the signature in the community database **7** may then be marked as safe rather than unsafe. Many other variations of monitoring safety may be done within this concept.

The database **7** may further include a behaviour mask for the object **4** that sets out the parameters of the object's performance and operation. If an object is allowed to run on a remote computer **2**, even if the initial signature search **22** indicates that the object is safe, then operation of that object may be monitored within the parameters of the mask. Any behaviour that extends beyond that permitted by the mask is identified and can be used to continually assess whether the object continues to be safe or not.

The details of an object **4** that are passed to the base computer **3** may be in the form of a signature or "key" that uniquely identifies the object **4**. This is mainly to keep the data storage and transmission requirements as minimal as possible. This key may be formed by a hashing function operating on the object at the remote computer **2**.

The key in the exemplary embodiment is specially arranged to have at least three severable components, a first of the components representing executable instructions contained within or constituted by the object, a second of the components representing data about the object, and a third of the components representing the physical size of the object. The data about the object in the second component may be any or all of the other forms of identity such as the file's name, its physical and folder location on disk, its original file name, its creation and modification dates, resource information such as vendor, product and version, and any other information stored within the object, its file header or header held by the remote computer **2** about it; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers. In general, the information provided in the key may include at least one of these elements or any two or more of these elements in any combination.

In one embodiment, a checksum is created for all executable files, such as (but not limited to) .exe and .dll files, which are of the type PE (Portable Executable file as defined by Microsoft). Three types of checksums are generated depending on the nature of the file:

Type 1: five different sections of the file are check summed. These include the import table, a section at the beginning and a section at the end of the code section, and a section at the beginning and a section at the end of the entire file. This type applies to the vast majority of files that are analysed;

Type 2: for old DOS or 16 bit executable files, the entire file is check summed;

Type 3: for files over a certain predefined size, the file is sampled into chunks which are then check summed. For files less than a certain predefined size, the whole file is check summed.

For the check summing process, in principle any technique is possible. The MD5 (Message-Digest algorithm 5) is a widely-used cryptographic hash function that may be used for this purpose.

This allows a core checksum to be generated by viewing only the executable elements of the checksum and making a comparison between two executables that share common executable code.

For the type 1 checksum mentioned above, three signature processes may be used. The first defines the entire file and will change with almost any change to the file's content. In particular, the first defines a representative sampling of the contents of the program in that if any fundamental change is made to the program, the checksum will change, but trivial changes can be identified as such, allowing correlation back to the original program body. The second attempts to define only the processing instructions of the process, which changes much less. The third utilises the file's size, which massively reduces the potential of collisions for objects of differing sizes. By tracking the occurrences of all signatures individually appearing with different counterparts, it is possible to identify processes that have been changed or have been created from a

common point but that have been edited to perform new, possibly malevolent functionality.

As well as checksum data, in many embodiments metadata about the object is captured and sent to the base computer **3**. Amongst other types, the types of metadata captured and sent to the base computer **3** might be:

"Events": these define the actions or behaviours of an object acting upon another object or some other entity. The event may include three principal components: the key of the object performing the act (the "Actor"), the act being performed (the "Event Type"), and the key of the object or identity of another entity upon which the act is being performed (the "Victim"). For example, the event data might capture the identity, e.g. the IP address or URL, of a network entity with which an object is communicating, another program acting on the object or being acted on by the object, a database or IP registry entry being written to by the object, etc. While simple, this structure allows a limitless series of behaviours and relationships to be defined. Examples of the three components of an event might be:

Actor	Event Type	Victim
Object 1	Creates Program	Object 2
Object 1	Sends data	IP Address 3
Object 1	Deletes Program	Object 4
Object 1	Executes	Object 2
Object 2	Creates registry key	Object 4

"Identities:" these define the attributes of an object. They include items such as the file's name, its physical location on the disk or in memory, its logical location on the disk within the file system (its path), the file's header details which include when the file was created, when it was last accessed, when it was last modified, the information stored as the vendor, the product it is part of and the version number of the file and it contents, its original file name, and its file size.

"Genesisactor": the key of an object that is not the direct Actor of an event but which is the ultimate parent of the event being performed. For example in the case of a software installation, this would be the key of the object that the user or system first executed and that initiated the software installation process, e.g. Setup.exe.

"Ancillary data": many events may require ancillary data, for example an event such as that used to record the creation of a registry run key. In this situation the "event" would identify the Actor object creating the registry run key, the event type itself (e.g. "regrunkey"), and the Victim or subject of the registry run key. The ancillary data in this case would define the run key entry itself; the Hive, Key name and Value.

"Event Checksums": because the event data can be quite large extending to several hundred bytes of information for a single event, its identities for the Actor and Victim and any ancillary data, the system allows for this data itself to be summarised by the Event Checksums. Two event checksums are used utilising a variety of algorithms, such as CRC and Adler. The checksums are of the core data for an event. This allows the remote computer **2** to send the checksums of the data to the central computer **3** which may already have the data relating to those checksums stored. In this case, it does not require further information from the remote computer **2**. Only if the central computer **3** has never received the checksums will it request the associated data from the remote computer **2**. This affords a considerable improvement in performance for both the remote and central computers **2**,**3** allowing much more effective scaling.

Thus, the metadata derived from the remote computers **2** can be used at the community database **7** to define the behaviour of a process across the community. As mentioned, the data may include at least one of the elements mentioned above (file size, location, etc.) or two or three or four or five or six or all seven (or more elements not specifically mentioned here). The data stored in the community database **7** provides an extensive corollary of an object's creation, configuration, execution, behaviour, identities and relationships to other objects or entities that either act upon it or are acted upon by it. This may be used accordingly to model, test and create new automated rules and heuristics for use in the community database **7** and as rules that may be added to those held and used in the local database of the remote computers **2** to identify and determine the response of the remote computers **2** to new or unknown processes and process activity.

Moreover, it is possible to monitor a process along with any optional sub-processes as a homogenous entity and then compare the activities of the top level process throughout the community and deduce that certain, potentially malevolent practices only occur when one or more specific sub-processes are also loaded. This allows effective monitoring (without unnecessary blocking) of programs, such as Internet Explorer or other browsers, whose functionality may be easily altered by downloadable optional code that users acquire from the Internet, which is of course the principal source of malevolent code today.

Distributed Architecture

The system as described so far is generally as described in our previous application US-A-2007/0016953. There now follows a description of new, advantageous schemes in operating such a system. It will be appreciated that in the description in relation to FIGS. **1** and **2** the central computer and the community database are presented for convenience as single entities. As will be apparent from the following discussion, the base computer **3** can be comprised of multiple computers and servers, etc. and the community database **7** can be made up multiple databases and storage distributed around this central system.

FIG. **3** shows an example of an arrangement for the base computer **3**. The remote computer **2** has an agent program **10**, which generally includes the same functionality as the agent program described above in relation to FIGS. **1** and **2**. In short, the agent program **10** the monitors objects on that computer **2** and their behaviour and communicates with the base computer **3** to send details of new objects and behaviour found on the remote computer **2** and to receive determinations of whether the objects are safe or not. The agent program **10** optionally communicates with a local database **11**, which holds a local copy of signatures relating to the objects found on the remote computer **2**.

The agent program **10** can be comparatively small compared with other commercially available anti-malware software. The download size for the agent program **10** can be less than 1 MB and occupy about 3 MB of memory when running, using more memory only to temporarily hold an image of a file(s) being scanned. In comparison, other anti-malware packages will typically have a download of 50 MB and occupy 50 MB to 200 MB of computer memory whilst scanning files. Thus, the exemplary architecture can occupy less that 2% of system resources compared with other products.

This is achieved primarily by the exemplary agent program **10** being developed in a low-level language, having direct access to system resources such as the video display, disk, memory, the network(s), and without the incorporation of many standard code or dynamic linked libraries to perform these functions. Memory usage is optimised by storing data in

the local database structure that provides the ability to refer to objects by a unique identifier rather than requiring full filenames or signatures. All unnecessary dynamic link libraries are unloaded from the process immediately as they are identified as no longer being used, and background threads are merged to reduce CPU usage. A small, efficient agent can be deployed more quickly and can used alongside other programs, including other security programs, with less load on or impact on the computer's performance. This approach also has the advantage of having less surface area for attack by malware making it inherently more secure.

The agent program **10** communicates with the base computer **3** over the Internet **1** via the Internet's Domain Name Resolution System (DNS) **50**.

The base computer **3** has a first layer comprising one or more servers **61**, which in this example are web servers. In the present disclosure, the web servers **61** are referred to as "FX servers" or "threat servers" and the first layer is referred to as the "FX layer". However, it will be appreciated that any type of suitable server may be used according to need. The remote computers **2** are allocated to one of the web servers **61** as explained in more detail below.

The FX layer **60** makes real time decisions as to whether or not an object is benign or malevolent based on the details of that object sent from the remote computer **2**. Each web server **61** of the FX layer **60** is connected to a database **62** which contains entries for all of the objects known to the base computer **3** and a classification of whether the object is safe or unsafe. The database **62** also stores rules for deciding whether unknown objects are safe or unsafe based on the information received from the remote computer **2**. The web server **61** first searches the database **62** for matching signatures to determine if the object is safe or unsafe or unknown. As described above, the signature can be a function of the hashes (checksum data) derived from file itself, event data involving the object and or metadata about the object in any combination. If the object is unknown, the web server **61** then determines if any of the rules classify the object as unsafe. The web server **61** then communicates back to the remote computer **2** whether the object is safe and can be allowed to run, unsafe and is to be prevented from running, or unknown in which case the object can be allowed to run or not according to user preference.

Thus, the FX layer **60** reacts in real time to threats to determine whether or not malware should be allowed to run on a remote computer **2**.

Sitting behind the FX layer **60** is a second layer (called the "ENZO layer" **70** herein). Information about objects received by the FX layer **60** from the remote computers **2** is sent to the ENZO layer **70** such that a master record is maintained at the ENZO layer **70** of all data received from all remote computers **2**. This process is described in more detail below.

The ENZO layer **70** has one or more servers **74** (referred to as "central servers" or "ENZO servers" in this disclosure) connected to one or more "master" databases which aggregate all of the information received from all web servers **61** from all remote computers **2**. The databases may comprise any combination of the following three databases and in one exemplary embodiment has all three.

1) An object database **71**, which has entries containing object signatures (e.g. their MD5 checksums) and including metadata received about objects, e.g. file name, file location, or any other metadata collected by the system as described above.

2) A behaviour database **72** for capturing details of the behaviour of objects observed on a remote computer **2**. This database **72** is populated by the event data sent from the remote computers **2** and allows a picture to be built up of the behaviour and relationships an object has across the community.

3) A computer-object database **73** which associates remote computers **2** in the community with objects observed on those remote computers **2** by associating objects with identification codes sent from the remote computer **2** during object detail transmission. The computer may be identified by at least the following three identifiers sent from the remote computer: one which relates to the physical system, one which relates to the operating system instance and one which relates to the logged on user.

As will be appreciated, in a community having for example 10 million or more remote computers **2**, each having on average hundreds to thousands of objects, each having a large number of behaviours and associations with other objects, the amount of data held in the databases **71**,**72**,**73** is enormous.

The ENZO servers **74** can query the databases **71**,**72**,**73** and are operable, either by automation or by input from a human analyst or both, in monitoring object behaviour across the community and in investigating objects and developing rules for spotting malware. These rules are fed back to the FX layer **60** and or the agent program **10** running on remote computers **10** and, as discussed elsewhere in this document, are used by the web servers **61** or the agent program **10** in real time to stop malware from running on remote computers **2**. The performance of those rules in accurately stopping malware on the remote computers **2** is fed back to the ENZO layer **70** and can be used to refine the rules. The techniques used are described further in the following disclosure.

In an exemplary embodiment, some or all of the servers **61**,**74** and databases **62**,**71**,**72**,**73** of the FX and ENZO layers **60**,**70** are implemented using cloud computing. Cloud computing is a means of providing location-independent computing, whereby shared servers provide resources, software, and data to computers and other devices on demand. Generally, cloud computing customers do not own the physical infrastructure, instead avoiding capital expenditure by renting usage from a third-party provider. Cloud computing users avoid capital expenditure on hardware, software, and services when they pay a provider only for what they use. New resources can quickly be put on line. This provides a large degree of flexibility for the user.

An example of cloud computing is the Amazon Elastic Compute Cloud (EC2), which is a central part of the Amazon.com cloud computing platform, Amazon Web Services (AWS). Another example is the Windows Azure Platform, which is a Microsoft cloud platform that enables customers to deploy applications and data into the cloud. EC2 is used in the present example to provide cloud computing. Nonetheless, it will be appreciated that, in principle, any suitable cloud architecture could be used to implement the base computer **3**. Alternatively conventional data centres could be used to implement the base computer **3**, or a mixture of conventional data centres and cloud-based architecture could be used.

EC2 allows users to rent virtual computers on which to run their own computer applications. EC2 allows scalable deployment of applications by providing a web service through which a user can boot an Amazon Machine Image to create a virtual machine, which Amazon calls an "instance," containing any software desired.

A user can create, launch, and terminate server instances as needed, paying by the hour for active servers, allowing the provision of "elastic" computing. EC2 provides users with control over the geographical location of instances, which allows for latency optimization and high levels of redundancy. For example, to minimize downtime, a user can set up

server instances in multiple zones which are insulated from each other for most causes of failure such that one backs up the other. In this way, the cloud provides complete control of a user's computing resources and its configuration, i.e. the operating system and software packages installed on the resource. Amazon EC2 allows the user to select a configuration of memory, CPU, instance storage, and the boot partition size that is optimal for the operating system and application. EC2 reduces the time required to obtain and boot new server instances to minutes, allowing the user to quickly scale capacity, both up and down, their computing requirements change.

EC2 maintains a number of data centres in different geographical regions, including: US East Coast, US West Coast, EU (Ireland), and Asia Pacific (APAC) (Singapore). As shown by FIG. **4**, in an exemplary embodiment, the servers and databases of the base computer **3** are implemented in plural of these geographical locations **80**.

The important question therefore arises of how to allocate remote computers **2** to a particular instance of web server **61** in the FX layer **60**. Important considerations in this decision are to minimise latency and to share the load evenly across the web servers. Ideally, the load on resources would be shared approximately equally across all instances of web servers in order to avoid web servers being under utilised and to minimise the number of web servers required. It is also desirable to implement a simple solution which does not add significantly to the speed or size of the agent program **10** on the remote computers, or place an overhead on the base computer **3**.

1) In an exemplary embodiment, the agent program **10** uses the geographical location of the remote computer **2** in allocating the remote computer **2** to a web server **61**, i.e. based on its proximity to a geographical region **80** of the cloud. The location of the remote computer can be determined by any convenient means, such as examining the IP address of the remote computer **2** against a database of known location ranges, or performing a network route analysis to determine the number of "hops" required to reach the destination server. This functionality can be performed by the agent program **10** running on the remote computer **2** when it is first installed on the computer **2**. And by allocating based on location, latency can be minimised.

2) In an exemplary embodiment, the agent program **10** generates a random number based on the date and time of its installation seeded with unique identifiers of the remote computer to decide to which web server **61** it should be mapped. This provides a simple, pseudo random way of mapping remote computers **2** to web servers **61**, which shares the load evenly between web servers **61** even if several thousand remote computers are being installed simultaneously.

3) The DNS distribution layer then maps the URL to the physical data centre. The Domain Name System (DNS) is a hierarchical naming system built on a distributed database for computers, services, or any resource connected to the Internet or a private network. It associates various information with domain names assigned to each of the participating entities. The DNS distribution network **50** allows load balancing to plural servers **61** making up the FX layer **60**.

Any combination of these techniques can be used by the agent program **10** to allocate a remote computer **2** to a web server **61**. And all three may be used in combination.

The agent program may also utilize a secondary algorithm to re-map the remote computer **2** to different URL in case the primary URL is unavailable for any reason, e.g. if the servers go down. Also, in the event that malware has been able to disable the DNS mechanism then the agent program **10** can make direct contact with the FX servers **61** by referring directly to a series of direct IP addresses reserved for this purpose.

Optionally, the agent program **10** is arranged so that it can reassign the remote computer **2** to a different server **61** by receiving feedback/instruction from the server **61** the computer is already connected to. This can be used to add central control to the allocation of agent program **10** to servers **61**, which adds more flexibility to the system.

Thus, this scheme allows a simple, agent managed way of dynamically load balancing between servers and providing resiliency, whilst dealing with geographical location. In known prior art arrangements, specialised hardware load balancers are employed in the network to divide load to other servers, or else agents are manually allocated to web servers **61** when the agent program **10** is installed.

Using a cloud-based distributed architecture for the base computer **3** presents challenges in trying to maintain integrity of data across the servers and in managing the update of data across the many servers. As will be clear from the foregoing, the web servers **61** in a typical implementation will be dealing with huge amounts of data. There will be large amounts of commonality in the data, which can be used for determining rules on what objects are malware. However, it is impractical to store all data on all web servers **61** on all regions. This presents a problem in determining whether or not the data is common in real time.

To address these issues, an exemplary embodiment adopts the following scheme. Each web server **61** in the FX layer **60** examines an incoming packet of data from a remote computer **2** about an object seen on that remote computer **2**. The agent program **10** generates a checksum of the data packet and sends it to the web server **61**. The web server **61** has a database **62** of objects that have previously been seen and which have already been sent to the ENZO server **74**. The web server **61** checks its database **62** for the checksum of the incoming data. If the checksum is found, then the data has already been seen by the ENZO server **74**. In this case, the web server **61** just increases the count associated with that checksum in its database of the number of times that data has been seen to assist in determining the popularity of a piece of software or the frequency of the occurrence of a specific event. This information can then be forwarded to the ENZO server **74**. If the checksum is not found in the FX database **62**, then the web server requests the full data packet from the remote computer and then forwards this to the ENZO server **74**. The ENZO server **74** then stores this data in the appropriate databases **71**,**72**,**73**. Thus, the ENZO layer **70** keeps a master list in the object database **71** of all data objects, their metadata and behaviour seen on all of the remote computers **2**. This master list is propagated back to all of the web servers **61** in all of the geographical locations **80**. Thus, all web servers **61** are updated with information about which objects have been seen by the community.

This scheme has been found to reduce workload and traffic in the network by a factor of about 50 compared with a conventional scheme where typically each piece of data on being received by a web server would immediately be propagated to all other servers in the system.

It is desirable that the data held in the master databases **71**,**72**,**73** in the ENZO layer **70** provide scalability, accessibility and resilience to the data for users. It will be appreciated that if just one live ENZO server **74** is used, then if that server becomes inaccessible, all live data is lost. For this reason, in many embodiments at least one ENZO server **74** resides in each region **80** of the cloud to provide redundancy. This means that the web servers **61** are not all directly linked to the

same ENZO server **74**, but rather the web servers **61** are only linked to the ENZO server **74** in their own region **80**. This creates a need for an extensible method of providing data to multiple database servers.

FIG. **5** shows schematically a scheme of updating data from a web server **61** to an ENZO server **74**. Each web server **61** has two temporary databases **63***a*, **63***b* linked to it. At an initial starting point, one database **63***a* is part full **63***b* (active) and the other is empty (inactive). As information about objects is received from a remote computer **2** by the web server **61**, it posts the information into the active database **63***a*. This information will be checksum information received from the remote computers **2** about objects or events seen on the remote computer. If the checksum is previously unseen at the web server, the associated data about that object or event is posted to the database along with its checksum.

Once the active database **63***a* reaches a predetermined size or some other predetermined criteria is met, the contents of the active database **63***a* are put into long term storage **66** together with a time stamp of the point in time when this happens. While this is occurring, the inactive database **63***b* is made active and starts to fill with new data as it arrives, and the formerly active database **63***a* is cleared and made inactive. Again, when this database is full, then the process of time stamping and moving the contents of the database to long term storage is repeated and the databases are swapped over. In some embodiments, the FX layer **60** has a S3 sub-layer **65**, which is responsible for managing input/output to the long term storage **66**. S3 (Simple Storage Service) is an online storage web service offered by Amazon Web Services. S3 is an Amazon service for the storage of high volume data in a secure and non-volatile environment through a simple web services interface. S3 is used to store the history of the FX server data being sent from the FX servers to the ENZO servers. When a block of data is put into long term storage **66**, the S3 layer **66** also forwards this data to the ENZO server **74** or servers so that the ENZO server can update its databases. All FX servers feed all ENZO servers via the S3 storage service.

The databases **63***a*,**63***b* may be emptied into long term storage **66** for example every few minutes or even every few seconds. However, as will be appreciated, the actual time period used can be selected according to the load and size of the databases and may be chosen dynamically based on for example the load on the database or the size of the data. Thus, the S3 layer **65** holds all of the data received by all of the web servers **61** in the community. This has a number of advantages as illustrated by FIG. **6**.

Firstly, this can be used to allow the master ENZO databases **71**,**72**,**73** to be rebuilt for example in the event that an ENZO server **74** develops a problem. Also, if the data held by an ENZO server **74** is corrupted somehow, for example by a problem in a software upgrade on the ENZO server **74**, it is possible to roll back the databases at the ENZO server **74** to a point before the corruption occurred. By using the timestamps associated with the blocks of data in the S3 layer **65**, the ENZO server **74** can request the data blocks in long term storage **66** at the S3 layer **65** to be resent to the ENZO server **74** so the server's databases **71**,**72**,**73** can be brought up to date.

Furthermore, using this scheme, the ENZO servers **74** in multiple regions **80** can create backup databases of the ENZO databases by receiving the data from the S3 layer. This can be done without affecting the live ENZO databases **74**, so these can carry on functioning seamlessly.

The alternative would be to post to databases at a single ENZO server which is designated the master server and to propagate the data from the master server to other servers in the community. However, this in effect creates one "live" server and one or more backups. The live server therefore does most of the work, e.g. pre-processing data received from the agents, updating the databases and analysing the data for threats, whilst the backup servers are under utilised. Thus, the live server must be more powerful than the backup servers. This tends to be wasteful and inefficient of resources. In comparison, the exemplary embodiment, shares the workload around the servers equally.

It is also desirable for operators, human malware analysts, etc., to have a copy of the data from a live server for research, etc. Image servers may be created in the same way as the live servers and the backup servers. The user of the system can take the image server offline when he needs to take an image for offline use. Once the image is taken, the image database can be brought up to date using the blocks of data from the S3 layer.

Cloud computing can recreate instances of web servers very quickly, e.g., in a matter of minutes. Cloud computing is therefore advantageous in implementations where the volume of transactions grows (or shrinks). In this situation, a new server instance can be quickly created to handle the new load. However, it is necessary to take an image of a live server to create the image. This is disruptive to the processes running on that instance during imaging. It is therefore clearly not acceptable to take a live server out of commission for this period. Therefore, in many embodiments, a special dedicated image server is maintained behind the ENZO server, which is updated in a similar way to the live and backup servers. In this case, the image server is taken off-line and imaged. This process might take for example 30 minutes. During this time, the image server is not receiving updates from the S3 server and therefore becomes out of date. However, since the exact time when the server went off line is known, it can be updated by requesting data blocks from the S3 server with a timestamp later than when it went off line. Thus, an image can be taken of an updated server for offline use etc. and brought back up to date without affecting the running of the system.

It is also desirable to be able to scale out the architecture of the ENZO layer **70**. For example, a live server **74** may be running out of processing power due to the increase in traffic it is seeing from web servers **61** or agents **10**. For example, more remote computers **2** may be connected to the system, or a common software package may be updated leading to a large volume of new data being sent to the ENZO layer **70**. In this situation, a second or further live server instance can be created in the ENZO layer **70** in order to scale out horizontally (as opposed to adding more CPU/RAM to an existing server). As described above, it is relatively straightforward to create an image of an ENZO server **74** in a cloud-computing environment. As can be seen from FIG. **7**, the data on each server **74** is divided into a plurality of sections. In this case the data on the server **74** is divided into eight sections **0**-**7**. Each section does a share of the processing on that server. When the server is divided, various sections are allocated to each server. So, for example, in the example of FIG. **7**, the original server L**1** processes data **0**-**3**, and the newly created server L**2** processes data **4**-**7**. Each server then handles pre-processing and storing its own data. For the purposes of analysing the data, in many embodiments the data from these multiple severs is aggregated so that the human analyst is working on the complete set of data when investigating malware (as described in more detail below). Of course, the data can be divided in different ways among the servers according to need.

So, instead of holding a single database and then having to dissect it if it is required to distribute the load across multiple

servers, in many embodiments the database is already dissected making it is much easier to make a transition to further distribution of the database workload across multiple servers.

The number of data sections may be selected to be eight, because this is enough to give reasonable flexibility and ability to expand. However, it is rare that a business needs to expand by more than 800% at any time. Nonetheless, other plural numbers of data sections can be used as appropriate.

This architecture allows vast amounts of data to be managed across multiple locations and servers. The architecture can be scaled and servers copied without incurring extra work or affecting the running of the system. The data can be exchanged across servers that can change dynamically.

Using Trend Data

Malware evolves at a significant pace. Historically, malware used a single infection (module or program) to infect many computers. As this allowed malware researchers to quickly become aware of the infection, signatures could be quickly deployed to identify and block the threat. In the last decade, malware authors have made their threats contain many varied modules, which shared a common purpose. To detect and block these new threats is a much greater challenge for malware research and to creating and distributing signatures to identify and block all of the variations, any one of which could infect one or more computers. Due to the varied and distributed nature of these modern threats malware researchers are often late in identifying that a threat even exists.

By collecting hashes, signatures and statistics of new programs and modules correlated by path and file details, and aggregating this information at a central location, it is possible to use a method of analysis that will allow faster identification of new threats.

On any given day the amount of new commercial programs and modules that are created can be considered almost constant. This data can be measured against any of the metadata or event data collected about the object. In one example, the data is measured against the Path (Folder) location of new files created. The data can also be measured against their filenames, and file types (e.g. .exe, .dll, .drv, etc.) along with details such as vendor, product and version information, registry keys, or any other system derived data forwarded to base computer. Any combination of these factors is possible. In whichever way it is chosen to group the data, by establishing a baseline for the pattern and distribution of this data across the groups it is possible to quickly identify outliers.

FIG. **8** shows an example of collected data over a time period grouped according to folders in the file system compared with the established normal distribution (base line). If the number of new programs in the windows system32 subfolder is normally x in any given time period t (e.g., in a given day) and in a particular time period t**1** under consideration it is 1.4x (or exceeds some other predetermined ratio) then it indicates an abnormal increase in the number of new files which signifies possible malware activity. This identification allows greater focus to be placed on researching these outliers. Processes can be readily developed that can use this outlier information to automatically group, identify or prioritise research effort and even make malware determinations. Where automated rules are used to identify malware, this information can be used to automatically heighten sensitivity of these rules when applied to objects in the group that has been identified as an outlier.

Another example of this technique is to focus on a filename and compare the number of new objects (as represented by its hash) that utilize that name, or the number of new objects all of which have new names. This is contrary to the general principles of most commercial software, which needs consistency to enable manageability of their products. Most commercial applications have only a few names and each file name has only a few variants as each new release is made available. In contrast, some malware has thousands of variants using the same, similar or similarly structured name and which are observed by the community utilising the related names within a very short window of time (e.g., seconds-weeks).

In principle, any length of time for the window can be selected. In practice, this is likely to be of the order of every 15 minutes, or every few hours, or a day, rather than longer periods like every week.

In some embodiments, techniques are provided to provide context for researchers. In detecting malware, human analysts and computer automated analysis have different strengths and are often used together to some degree to harness the capabilities of both. Computers are good at applying rules to and processing large amounts of data. In the present application, the amount of data received from agent program **10** running on remote computers **2** is vast and therefore suited to being processed by computer. It would be impractical to have human operators to look at all of this data to determine malware. On the other hand, experienced human analysts are highly adept at spotting malware given the appropriately focussed data about that object and similar objects and their context. Often a human analyst can "intuitively" spot malware, in a way that is difficult to reduce to a set of rules and teach to a machine. For example, kernel32.dll is a file associated with the Windows® operating system. A malware program may be called something similar, e.g. kernel64.dll, which would immediately be apparent to a user as being suspicious, but which it would be difficult to teach a machine to spot.

The pattern may be obvious in retrospect when an analyst is presented with the relevant data. However, this is a rear view mirror approach. The problem is finding the relevant data in the vast amount of data being received at the base computer **3** so that the analyst can focus his/her time on the important data. As illustrated by FIG. **9**, what is needed is some what of distilling the raw data **100** into groups of related data **101** to provide a focussed starting point for a malware researcher to investigate and take necessary action **102** to flag objects bad or safe **103** accordingly.

As will be appreciated, a single object on a remote computer **2** across the community can have a high degree of associations with other objects on remote computers **2**. Some tools currently exist which can map links between entities to create a network of associations. However, when used for a malware object and its associations with other objects, these will result in a diagram that is so dense and complicated that it is practically useless for the purpose of allowing a human analyst to identify malware. The problem is sifting this data so useful features are extracted for the analyst so that he can quickly focus on the likely candidates for being malware.

An exemplary embodiment, illustrated by FIGS. **10** to **13**, provides a software program running on the ENZO server **74** and giving the user a visual based tool for running queries against the databases. A query language such as TSQL may be used for example to query the database based on the user's input. As mentioned in the foregoing, the ENZO layer **70** has a complete record of the properties and activities of all objects seen on all remote computers **2** in their databases. FIG. **10** shows an example of an on-screen display and user interface including a table **105** in which the rows represent the objects (determined by their signatures) that match the results of the current query. The user interface provides tools for creating

new queries **106**, for displaying current and recent queries **107** and for sorting the data displayed in the table for the current query **108**. Clearly there may be more results than will fit on one screen. There may therefore be more than one page of results to which the user can navigate through appropriate controls **109** provided by the user interface.

The table **105** has a plurality of columns of data representing information about the objects. The information may include any of the data collected about an object, its metadata or its behaviour, sent to the base computer **3** by the remote computers **2**. This may include any of the following for example: the name of the file, and the location (path) of the file, the country where the object was first encountered, the identity of the remote computer that encountered the object, its operating system and version number, details of the web browser and any other security products installed on the computer, the creator of the file, the icon associated with the file, etc. The information may include metadata about the file, such as the author or the size of the object or information relating to executable instructions in the object. The data may contain "actor" and "victim" information and details of its registry key. The information may include information about whether or not the object has been identified as safe or unsafe at the remote computer **2** or at the FX server **61**.

The information displayed may contain counts of how often a particular attribute is present among all the objects returned by the query or across the community. In addition, the information may contain less easily understood information. As discussed above, the information collected about an object can include checksums of parts of the executable, e.g. the first 500 bytes of the executable part of the file. In many cases, the actual value of the checksum or other data is of little or no interest to the user. Therefore, in many implementations some or all of this information is presented to the user in a graphical way, utilising colour, shape and or different symbols, which makes spotting patterns in the data more intuitive and speedy for the user.

In some embodiments, at least one column of data is presented according to the following scheme:

1) if the value is unique in that column between the rows of data representing objects then the numeral "1" is presented.

2) if the value is replicated between the rows of data on the screen, or between all the rows of data matching that query, then a coloured symbol (e.g. a green triangle or a red letter "P" in a blue square, etc.) is allocated for that value and displayed in that column for all rows that share that value. Thus, without needing to know what the symbols represent, the user can see at a glance which rows on the screen have commonality of an attribute.

This provides a starting point for running further queries. For example, the user can quickly see commonality between objects, which look interesting. As explained elsewhere in this document, a typical trait of malware is that the objects use a large number of file names, file locations, similar file sizes, similar code portions, etc. The program allows a user to refine a query by for example clicking on columns in the table and using the query creating tools **106** to add a particular attribute value to the predicate of the query, or to start a new query with this column value in the predicate, etc.

The ultimate aim of this process is for the user to find a query that returns the maximum number of rows with commonality. By this process, the user identifies a set of criteria made up of hashes and other metadata definitions that can be used to accurately identify similar objects that share criteria. For example, FIG. **11** shows a user query that returns rows with a high degree of commonality representing objects that have commonality. The software also in many implementations allows the user to bring up a screen of statistics for the objects that have been returned from the query. In particular, as shown by FIG. **12**, the interface can display to the user the number of objects returned by the query that are known to the database as being malware, the number known as being not malware, and the number where it is not yet known whether or not the object is malware (represented in FIG. **12** by differently shaded/coloured bars). Other data about the results of a query or objects having a certain attribute selected by the user can be shown in graphical form, e.g. a bar graph, or a pie diagram as shown in FIG. **13**. The user can take this information into account as well as information shown in the table **105** when deciding whether or not the results of the query determine malware. The user can then take the appropriate action, such as tagging the objects returned by the query as malware, not malware, or suspicious and requiring further investigation. Thus, by being able to query and analyse the collective view of an object, i.e. its metadata and behaviours, across all agents **10** that have seen it, a more informed view can be derived, whether by human or computer, of the object. In addition it is possible to cross-group objects based on any of their criteria, i.e. metadata and behaviour.

To take an example, it might be a previously identified trait of certain types of malware that the malevolent object is located in the "recycle bin" of the operating system. This gives a starting point for the user to run a query to find all objects, or all objects encountered during a user specified time frame, that are located in the recycle bin by looking at their "path" attribute. This would display rows to the user corresponding to the different objects found (e.g. each row corresponding to a different MD5 signature). The user can then drill down through the rows by selecting interesting attributes by clicking on a value in a column. For example, the user may choose objects whose attributes show that the object's icon is the same as the icon used the "Notepad" program (which comes with Microsoft Windows® operating systems), but which are not Microsoft originating programs. In many implementations the system allows the user to apply Boolean operators in refining the query. The system allows the user to bring up statistics of how well the query cover objects that are known by the system to be malevolent or benign to give the user more information about useful the query is in classifying objects.

The users employed in grouping objects according to this scheme need not be highly skilled malware analysts. The users need only basic training in use of the tool and in grouping objects by running queries and looking for commonality. The final analysis of objects covered by the query can be taken by a skilled malware analyst. The advantage of this system is that the computer is doing the work in processing the raw data which is too large of a task to be practical for a human operator to complete. The human operators grouping the data need not be skilled, which reduces costs for the business. This reduces costs and leaves the skilled malware analysts to concentrate on queries that are found to be good candidates for further investigation. Thus, the skill of these operators is most effectively deployed in using their experience to identify malware.

Referring to FIG. **9**, if at stage **103** the user decides that a query (Q**5**) is deterministic in classifying the objects returned by a query as safe or unsafe, then the rule can be passed back to the web servers **61** and or the agent program **10** at the remote computers **2** and used in real time in deciding whether new objects seen at the web servers **61** or remote computer **2** are safe or unsafe.

If a new object that has not been seen before by the community is received by a remote computer **2** or a web server **61**

and is deemed by one of the rules run by the remote computer **2** or web server **61** to be unsafe, the object can be stopped from running on the remote computer **2** in real time, or the user of the remote computer **2** can be warned and given the option of continuing or not. Furthermore, the remote computer **2** (via the web server **61** to which it is assigned) or the web server **61** can report to the ENZO server **74** that a new object has been determined to be safe or unsafe by the rule. The ENZO server **74** can update its master database with this finding and this finding can be communicated to other remote computers **2** and or web servers **61** and used to update their databases that objects with that signature are unsafe and should be blocked in the future, or safe and should be allowed to run.

Alternatively, the rule can be used by human analyst at the ENZO layer to monitor suspicious objects. Objects covered by the rule can be displayed to a human analyst and the analyst can then decide for each object in turn whether it is malevolent or benign, and mark the object accordingly. These determinations for each object are sent back to the web servers **62** and or agent software **10** and used to decide whether or not the object is allowed to run when encountered at the remote computers in real time. This method does not react as quickly to new threats, compared with applying the rules in real time at the web servers **62** and or agent software **10**. However, it may be preferred in cases where it is desired to keep the number of "false positives" low, and or where a rule is found to be good at identifying suspicious objects which need further investigation before a firm determination can be made that they are malware.

Furthermore, this action can be semi automated. The application can monitor the groupings along with any and all actions taken such as groups being determined as malicious or as benign. By tracking these actions it is possible for the ENZO server **74** to learn from human analysts how to identify and determine objects automatically. The application can remember queries that have been run by users. If the same query is being run repeatedly by a user and returning malware objects, or if a researcher consistently takes the action of determining the objects to be malware, then the system can automatically identify this condition and create a rule of the criteria and the action to determined matching objects as malicious. This rule can be sent to the FX server layer **60** to enable real time actions to be made to catch objects that match the rule and determine them as malicious. The identity of these rules and of malware grouping applied by the rules can then be fed back to both the agent and the ENZO server **74**. In this way the FX server **61** is performing an action sanctioned and distributed by the ENZO server **74**, on its behalf. Similarly, these same processes can be applied to benign groups of objects resulting in rules that determine such objects as "good" which can be propagated back to the FX layer **60** in the same way for the rules for "bad" objects. Thus, the ENZO layer **70** allows the user to easily identify these groupings. The ENZO layer can then monitor the groupings along with any and all actions taken such as groups being determined as malicious or as benign. By tracking these actions it is possible for the ENZO layer **70** to learn from humans how to identify and determine objects automatically.

The ENZO layer **70** can also monitor trends and bring information to the human analysts' attention, such as the evolution in the behaviour of a previously known good program. For example, a legitimate version of the game Solitaire will be observed by the ENZO servers to have a particular behaviour profile from the actions performed by that program on remote computers over a period of time. If after a period of time that program starts exhibiting new, suspicious behaviour, such as trying to install a driver or accessing a web server in another country, this can be flagged for investigation.

Furthermore, an exemplary embodiment of the agent program **10** at the remote computer provides an interface that allows the user to report "false positives" to the system, i.e. where an object has been flagged as malware, which the user believes to be legitimate software. The system is arranged to feedback this information to the ENZO server **74**. At this point, the system can determine that the rule is no longer reliably deterministic in finding malware and can disable the rule. Alternatively the system can flag the rule for manual review and or modification by a human analyst. Alternatively, it can be decided that the rule is generally useful despite the false positive and that the false positive is an outlier. The conflicting benign object can then be specifically excluded from the rule at the FX layer **70** by placing it on a white list.

It should be noted that rules can be prioritised (applied in order by priority) and can be established to only apply against objects having a specific determination, location, popularity, age or any other criteria known by or derived by the database. For example, if a first rule determines the object originates from Microsoft, then it may be decided not to apply any further rules. In contrast, if a first rule determines that an object is suspicious, then it may be decided to apply further rules as appropriate.

Thus, many embodiments of the software application running on an ENZO server **74** provide powerful object behaviour monitoring and assessment. The software allows the user to obtain a thorough understanding of an object's behaviour on a remote computer and the contextual environment in which it is observed, in part through providing powerful data visualisation techniques. The software enables large research teams to work collaboratively and incrementally and leverages human research skills and experience through expert system automation.

Configurable Firewall

It is desirable to stop malware before a computer becomes infected, e.g., when the infected file is received by email, downloaded, or when it is installed or run for the first time.

A well known example of a malware program is the "Zeus" Trojan. This program runs in the background on an infected computer and steals login details as they are entered on screen by a user by monitoring the user's keystrokes. This information is then sent to the criminal's servers where it can be used to take control of the infected computer. An instant message is sent to the criminal over the network when the malware program detects logon information being entered on an online banking website, so the criminal receives this information in near real time. The Trojan can then take remote control of the user's computer using the Virtual Network Connection protocol, a protocol that allows a remote user to access and take control of a computer remotely. The criminal can then steal money from the victim's bank account.

While most anti-malware software attempts to stop the malware from infecting a host computer or stopping communications with the malware as soon as possible, not much attention has been shown to finding and prosecuting the criminals responsible. One difficulty is that it is illegal for the anti-malware or even the police to hack into the criminal's servers to find the criminal's data. Indeed, it a known ruse for the criminals to use so-called "botnets", which are "zombie networks" of legitimate users' computers which have been infected by malware and are thereby under the control of the criminals.

An exemplary embodiment illustrated by FIG. **14** implements a firewall **95** with the agent program **10** at the remote computer **2**. A firewall per se is well known. However, nor-

mally firewalls are configured by rules input by the user to govern what programs have permission to access the network for incoming/outgoing traffic. The exemplary embodiment additionally takes rules from the base computer **3** to limit, which objects on the remote computer **2** can communicate across the firewall and with which entities **96**,**97** on the network. For example, information can be sent from the base computer **3** to the firewall **95** that known active URLs of Zeus servers **97** are blocked from exchanging messages from all objects on the remote computer **2**. Thus, the firewall **95** can implement a rule blocking communications with the criminal's servers.

This provides a valuable second line of defence against malware. It is generally desirable to stop malware at the point of infection by preventing the malware from infecting a computer. The next line of defence is to stop the malware from communicating. The configurable firewall **95** can therefore leverage the rules and knowledge developed about malevolent objects and network URLs held at the base computer **3** to provide a next line of defence in fighting malware.

Furthermore, in one embodiment, the agent program **10** can be used to contact the authorities **98**, e.g., a law enforcement agency, if malevolent Internet traffic is detected. The IP traffic between the infected computer **2** and the criminal's server **97** can be intercepted by the firewall **95**, stored and sent to the authorities' server **98**, so the authorities can see what information the criminals were stealing through the malware. This information can be sent to the authorities' servers **98** in encrypted form for added security. In the exemplary embodiment, checksums are made and stored of the data that sent to the authorities' servers **98**, as there are very strict processes in place for forensic data handling and evidence retention that should be observed.

Legal constraints will mean in most instances it is necessary to obtain permission from the user of the infected computer **2** before this information is captured and sent to a third party. In this situation, the agent program **10** will open a dialogue with the user to obtain permission before proceeding. In some instances, the user may be happy to give blanket permission for data to be sent whenever a malware program or a particular class of malware program is detected on his computer **2**, so it will not be necessary to obtain specific permission every time malevolent objects or communication is detected.

Self Defence

Malware has evolved to the state where it will attack, disable or otherwise destroy security products installed on a computer. There are two principle vectors, which need to be considered.

A first approach is to target the methods used by malware to identify security products. In this specific case the malware will use behaviour monitoring of processes running on the remote computer that view, observe, record, examine or otherwise interfere any aspect of the malware code. Examples might be a malware entity watching for processes or tasks which look at the registry keys the malware has used to configure, control or record its operation. As will be appreciated, malware is by its nature surreptitious. From the malware's point of view, no processes on the remote computer should be monitoring or interacting with the malware object, since the existence of the malware object on the local computer should be unknown. If any processes are monitoring the malware, it is therefore likely to be security software or a system monitoring tool or a system tool being used by the user of the computer **2** to rid the computer **2** of the infection.

Thus, by finding processes that are doing this, the malware process can find the identity of the process and its location or locations on disk. This identification then allows it the ability to delete, disable or interfere with the process or processes it has identified as potential threats. In most cases these will indeed be security programs or system monitoring tools, both which it might consider threats.

Many embodiments guard against this possible interference by malware programs by preventing the behavioural detection of a security program when examining the malware. This is achieved by hiding behind, masquerading as, and/or leveraging core operating system components to examine the malware's state and components. In particular, the agent program **10** performs operations by impersonating the system's context when it is necessary to rely on standard system calls, or circumventing any system calls by accessing data at the raw level as it is stored on disk. For example, the agent program **10** can directly impersonate the kernel of the operating system when calling the ZwTerminateProcess function to terminate a malevolent process. In doing so, the malware cannot trace the call back to the agent program. The malware considers this behaviour as normal, as it expects the operating system to interact with it (e.g. it is necessary for the operating system to be able to terminate the process when the computer is shut down), and so the malware does not attempt to terminate the agent program process. Furthermore, the malware cannot react by killing the process as the process is only identifiable as the operating system itself. To kill the operating system would also prevent the malware from operating.

FIG. **15** shows a hierarchy of control on a computer system **2** going from the kernel level **92** at the lowest level, the operating system level **91**, and a web browser running **92** on the operating system **91**. A second consideration is that, for malware to assert maximum control over the operation of the computer, it will often attempt to install a driver, possibly a kernel mode driver, which has complete control over processes and actions performed. A type of malware commonly known as a "rootkit" operates in this way and can, among other things, be used to conceal other malware, notably password-stealing key loggers and computer viruses. In certain cases the malware will load a malicious driver and use this to disable security applications and drivers from starting when Windows starts. In particular, the malware may have a list of drivers or program components that it knows correspond to security products.

To prevent malware from using this approach to disabling security applications the agent program **10** dynamically creates a second or backup driver or component **94** which loads as a backup of main component **93** during installation or bootup in the event that the main components **93** are inaccessible (as shown by FIG. **14**). It would have identical functionality but would be modified so as to appear different by name and form. This can be randomised, so that the malware cannot learn to identify the security program. In this scenario should the primary component fail to load, most likely due to being killed by malware, the second driver **94** would be loaded to perform its function. If the primary component **93** loads successfully, the secondary component **94** would simply terminate to avoid conflict by identifying the presence of the primary component **93**. By randomising the filenames and checksums of the installed objects comprising the agent program **10**, the agent program **10** is able to evade heuristic detection by active threats that would attempt to terminate or remove critical agent components.

Later versions of Microsoft's Windows operating system provide enhanced security by requiring system and low level drivers to be signed. This policy, called the kernel mode code signing policy, disallows any unauthorized or malicious driver to be loaded. Some malware creations modify the

master boot record to nullify driver certification checks. The malware executed by an infected master boot record bypasses driver signing policy by changing the boot options of Microsoft boot programs that will allow an unsigned driver to load.

So, in these conditions the secondary driver or component could remove its own digital signature to evade detection by the malware. This means it would fail to load unless the master boot record had been modified to disable driver verification. So it represents no threat to an uninfected system and would never load unless the system was infected and the master boot record was modified.

Embodiments of the present invention have been described with particular reference to the examples illustrated. However, it will be appreciated that variations and modifications may be made to the examples described within the scope of the present invention.

Those of skill in the art would understand that information and signals may be represented using any of a variety of different technologies and techniques. For example, data, instructions, commands, information, signals, bits, symbols, and chips that may be referenced throughout the above description may be represented by voltages, currents, electromagnetic waves, magnetic fields or particles, optical fields or particles, or any combination thereof.

Those of skill would further appreciate that the various illustrative logical blocks, modules, circuits, and algorithm steps described in connection with the embodiments disclosed herein may be implemented as electronic hardware, computer software, or combinations of both. To clearly illustrate this interchangeability of hardware and software, various illustrative components, blocks, modules, circuits, and steps have been described above generally in terms of their functionality. Whether such functionality is implemented as hardware or software depends upon the particular application and design constraints imposed on the overall system. Skilled artisans may implement the described functionality in varying ways for each particular application, but such implementation decisions should not be interpreted as causing a departure from the scope of the present invention.

One of ordinary skill in the art will appreciate that the various illustrative logical blocks, modules, and circuits described in connection with the embodiments disclosed herein may be implemented or performed with a general purpose processor, a digital signal processor (DSP), an application specific integrated circuit (ASIC), a field programmable gate array (FPGA) or other programmable logic device, discrete gate or transistor logic, discrete hardware components, or any combination thereof designed to perform the functions described herein. A general purpose processor may be a microprocessor, but in the alternative, the processor may be any conventional processor, controller, microcontroller, or state machine. A processor may also be implemented as a combination of computing devices, e.g., a combination of a DSP and a microprocessor, a plurality of microprocessors, one or more microprocessors in conjunction with a DSP core, or any other such configuration.

The steps of a method or algorithm described in connection with the embodiments disclosed herein may be embodied directly in hardware, in a software module executed by a processor, or in a combination of the two. A software module may reside in RAM memory, flash memory, ROM memory, EPROM memory, EEPROM memory, registers, hard disk, a removable disk, a CD-ROM, or any other form of storage medium known in the art. An exemplary storage medium is coupled to the processor such the processor can read information from, and write information to, the storage medium. In the alternative, the storage medium may be integral to the processor. The processor and the storage medium may reside in an ASIC. The ASIC may reside in a user terminal. In the alternative, the processor and the storage medium may reside as discrete components in a user terminal.

The previous description of the disclosed embodiments is provided to enable any person skilled in the art to make or use the present invention. Various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without departing from the spirit or scope of the invention. Thus, the present invention is not intended to be limited to the embodiments shown herein but is to be accorded the widest scope consistent with the principles and novel features disclosed herein.

What is claimed is:

1. A method of classifying a computer object as malware, the method comprising:

receiving, at a first threat server, details of a first computer object from a first remote computer, wherein the details of the first computer object include data uniquely identifying the first computer object;

determining, by the first threat server, whether the first computer object has been previously seen by comparing the data uniquely identifying the first computer object to a plurality of data uniquely identifying plural computer objects in a first database associated with the first threat server;

receiving additional information about the first computer object from the first remote computer when the first computer object has not been previously seen;

storing the details of the first computer object and the received additional information about the first computer object in a second database associated with the first threat server when the first computer object has not been previously seen;

providing contents of the second database to at least one database associated with a central server, wherein the contents comprise a signature of the first computer object, behavior information about the first computer object, and information about the first remote computer;

increasing a count associated with a number of times that the first computer object has been seen, and providing the increased count associated with the number of times that the first computer object has been seen to the central server; and

receiving, at a second threat server, at least a portion of the contents of the at least one database associated with the central server, wherein the at least a portion of the contents of the at least one database associated with the central server include a subset of the details of the first computer object stored in the second database.

2. The method according to claim **1**, further comprising storing at intervals, the contents of the second database in storage together with a timestamp and clearing the second database.

3. The method according to claim **2**, further comprising creating a backup central server by receiving at a second central server, all of the time-stamped blocks of data from the second database and incorporating all of the time-stamped blocks of data into at least one database associated with the second central server.

4. The method according to claim **2**, further comprising:

taking the central server off-line for a period of time such that the central server does not receive data from the first and second threat servers during that period of time;

after the period of time has elapsed, updating at least one database associated with the central server with time-stamped blocks of data from the storage that have a timestamp later than the time when the central server went off-line; and

bringing the central server back on line.

5. The method according to claim **2**, comprising:

rolling back at least one database associated with the central server to a point of time in the past;

updating the at least one database associated with the central server with time-stamped blocks of data from storage that have a timestamp later than the past point of time; and

bringing the central server back on line.

6. The method according to claim **2**, wherein the central server comprises:

a) an object database storing object signatures and metadata about objects;

b) a behavior database storing object behavior information; and

c) a computer-object database storing information about what objects are present on what remote computers.

7. The method according to claim **6**, wherein the threat and central servers are implemented using cloud computing.

8. The method according to claim **1**, further comprising:

receiving, at the second threat server, details of a second computer object from a second remote computer, wherein the details of the second computer object include data uniquely identifying the second computer object;

determining, by the second threat server, whether the second computer object has been previously seen by comparing the data uniquely identifying the second computer object to a plurality of data uniquely identifying plural computer objects in a third database associated with the second threat server;

determining that the second computer object has been seen before;

increasing a count associated with a number of times that the second computer object has been seen and providing the increased count associated with the number of times that the second computer object has been seen to the at least one central server; and

receiving, at the first threat server, a count associated with the number of times that the second computer object has been seen.

9. A system for classifying a computer object as malware, the system comprising:

a first threat server arranged to receive details of a computer object from a first remote computer, wherein the details of the first computer object include data uniquely identifying the first computer object, wherein the first threat server is further arranged to receive the details of the computer object from the first remote computer and determine whether the first computer object has been previously seen by comparing the data uniquely identifying the first computer object to a plurality of data uniquely identifying plural computer objects in a first database associated with the first threat server, wherein the first threat server is further arranged to receive additional information about the first computer object from the first remote computer when the first computer object has not been previously seen, store the details of the first computer object and the received additional information about the first computer object in a second database associated with the first threat server when the first computer object has not been previously seen, provide contents of the second database to at least one database associated with a central server wherein the contents comprise a signature of the first computer object, behavior information about the first computer object, and information about the first remote computer, and increase a count associated with a number of times that the first computer object has been seen;

the central server arranged to receive the increased count associated with the number of times that the first computer object has been seen; and

a second threat server arranged to receive at least a portion of the contents of the at least one database associated with the central server, wherein the at least a portion of the contents of the at least one database associated with the central server include a subset of the details of the first computer object stored in the second database.

10. The system according to claim **9**, wherein the first and second threat servers are arranged to store, at intervals, the contents of the second database in storage together with a timestamp and clear the database.

11. The system according to claim **10**, further comprising:

a backup central server having a database, the database of the backup central server being populated by receiving at the backup central server all of the time-stamped blocks of data from the storage and incorporating them into the database of the backup central server.

12. The system according to claim **10**, wherein, in the event that the central server is taken off-line for a period of time such that it does not receive updates of data from the first and second threat servers during that period of time, the central server is arranged to, after the period of time has elapsed, update at least one database with time-stamped blocks of data from the storage that have a timestamp later than the time when the central server went off-line.

13. The system according to claim **10**, wherein, in the event that at least one database of a central server is rolled back to a point of time in the past, the central server is arranged to update the at least one database with time-stamped blocks of data from storage that have a timestamp later than the point of time in the past.

14. The system according to claim **9**, wherein the central server comprises:

a) an object database storing object signatures and metadata about objects;

b) a behavior database storing object behavior information; and

c) a computer-object database storing information about what objects are present on what remote computers.

15. The system according to claim **14**, wherein the first and second threat servers and the central server are implemented using cloud computing.

* * * * *

Exhibit 8

US011409869B2

(12) **United States Patent**
Schmidtler et al.

(10) **Patent No.: US 11,409,869 B2**
(45) **Date of Patent: *Aug. 9, 2022**

(54) **AUTOMATIC THREAT DETECTION OF EXECUTABLE FILES BASED ON STATIC DATA ANALYSIS**

(71) Applicant: **Webroot Inc.**, Broomfield, CO (US)

(72) Inventors: **Mauritius Schmidtler**, Escondido, CA (US); **Gaurav Dalal**, San Jose, CA (US); **Reza Yoosoofmiya**, San Diego, CA (US)

(73) Assignee: **Webroot Inc.**, Broomfield, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **16/791,649**

(22) Filed: **Feb. 14, 2020**

(65) **Prior Publication Data**

US 2020/0184073 A1 Jun. 11, 2020

Related U.S. Application Data

(63) Continuation of application No. 14/709,875, filed on May 12, 2015, now Pat. No. 10,599,844.

(51) **Int. Cl.**

G06F 21/56 (2013.01)
G06N 5/04 (2006.01)

(Continued)

(52) **U.S. Cl.**

CPC ***G06F 21/565*** (2013.01); ***G06F 21/562*** (2013.01); ***G06N 20/00*** (2019.01);

(Continued)

(58) **Field of Classification Search**

CPC G06F 21/51; G06F 21/53; G06F 21/562; G06F 21/56; G06F 21/565; G06F 21/566;

(Continued)

(56) **References Cited**

U.S. PATENT DOCUMENTS

8,161,548	B1 *	4/2012	Wan	G06K 9/6228 726/22
8,510,836	B1 *	8/2013	Nachenberg	H04L 63/145 726/23

(Continued)

OTHER PUBLICATIONS

Bai, Jinrong, Junfeng Wang, and Guozhong Zou. "A malware detection scheme based on mining format information." The Scientific World Journal 2014 (2014). (Year: 2014).*

(Continued)

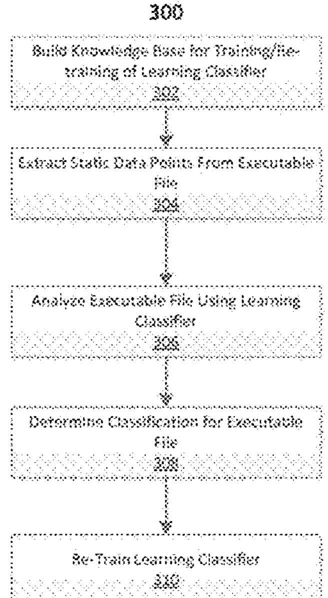
Primary Examiner — Michael Simitoski
(74) *Attorney, Agent, or Firm* — Sprinkle IP Law Group

(57) **ABSTRACT**

Aspects of the present disclosure relate to threat detection of executable files. A plurality of static data points may be extracted from an executable file without decrypting or unpacking the executable file. The executable file may then be analyzed without decrypting or unpacking the executable file. Analysis of the executable file may comprise applying a classifier to the plurality of extracted static data points. The classifier may be trained from data comprising known malicious executable files, known benign executable files and known unwanted executable files. Based upon analysis of the executable file, a determination can be made as to whether the executable file is harmful.

59 Claims, 4 Drawing Sheets

300



(51) **Int. Cl.**

G06N 20/10	(2019.01)
G06N 20/00	(2019.01)
G06F 11/30	(2006.01)
G06F 8/53	(2018.01)

(52) **U.S. Cl.**
CPC ***G06N 20/10*** (2019.01); *G06F 8/53* (2013.01); *G06F 2221/033* (2013.01)

(58) **Field of Classification Search**
CPC G06F 2221/033; G06F 8/53; G06N 20/00; G06N 20/10; H04L 63/145
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

8,719,924	B1	5/2014	Williamson et al.	
9,197,663	B1 *	11/2015	Gilbert	G06F 21/552
9,465,940	B1 *	10/2016	Wojnowicz	G06F 21/565
10,116,688	B1 *	10/2018	Yun	H04L 63/145
10,599,844	B2	3/2020	Schmidtler et al.	
2006/0026675	A1 *	2/2006	Cai	G06F 21/562 726/22
2006/0037080	A1 *	2/2006	Maloof	G06F 21/562 726/24
2007/0240220	A1	10/2007	Tuvell et al.	
2009/0013405	A1 *	1/2009	Schipka	G06F 21/562 726/22
2009/0274376	A1 *	11/2009	Selvaraj	G06K 9/6269 382/224
2009/0300765	A1 *	12/2009	Moskovitch	G06K 9/6256 726/24
2010/0082642	A1 *	4/2010	Forman	G06F 16/35 707/749
2010/0192222	A1	7/2010	Stokes et al.	
2010/0293273	A1 *	11/2010	Basarrate	H04L 61/2567 709/224
2011/0154490	A1 *	6/2011	DeLuca	G06F 21/56 726/23
2011/0172504	A1 *	7/2011	Wegerich	G16H 50/30 600/301
2012/0079596	A1 *	3/2012	Thomas	G06F 21/566 726/24
2012/0227105	A1 *	9/2012	Friedrichs	G06F 21/564 726/22
2013/0291111	A1 *	10/2013	Zhou	G06F 21/562 726/23
2014/0165203	A1	6/2014	Friedrichs et al.	
2014/0310517	A1 *	10/2014	Christodorescu	G06N 20/00 713/160
2015/0213365	A1 *	7/2015	Ideses	H04L 67/125 706/12
2016/0021174	A1 *	1/2016	De Los Santos Vilchez	H04W 12/128 709/201
2016/0156646	A1 *	6/2016	Hsueh	G06N 20/00 726/1
2016/0225017	A1 *	8/2016	Wong	G06Q 30/0247
2016/0335435	A1	11/2016	Schmidtler et al.	
2017/0262633	A1 *	9/2017	Miserendino	G06F 21/564
2018/0365573	A1 *	12/2018	Sultana	G06N 20/00

OTHER PUBLICATIONS

Office Action issued for U.S. Appl. No. 14/709,875, dated Dec. 8, 2016, 19 pages.

Shabtai et al., "Detection of Malicious Code by Applying Machine Learning Classifiers on Static Features: A State-of-the-Art Survery," Information Security Technical Report, Elsevier Advanced Technology, Amsterdam NL, vol. 14, No. 1, Feb. 1, 2009, pp. 16-29, XP026144093, ISSN: 1363-4127.

Office Action issued for U.S. Appl. No. 14/709,875, dated Mar. 24, 2017, 21 pages.

Office Action issued for U.S. Appl. No. 14/709,875, dated Apr. 20, 2018, 22 pages.

Office Action issued for U.S. Appl. No. 14/709,875, dated Jan. 24, 2019, 21 pages.

Office Action issued for U.S. Appl. No. 14/709,875, dated Jul. 17, 2019, 12 pages.

Bost, et al., "Machine learning classification over encrypted data." N DSS. vol. 4324. 2015. (Year: 2015), 14 pages.

Choi, et al., "Efficient Malicious Code Detection Using N-Gram Analysis and SVM," 2011 14th International Conference on Network-Based Information Systems. IEEE, 2011, 4 pages.

Devi, Dhruwajita et al., "Detection of Packed Malware," Proceedings of the First International Conference on Security of Internet ofThings, ACM, 2012, 5 pages.

Dube, et al., "Malware Target Recognition Via Static Heuristics," Computers & Security 31.1, 2012, 11 pages.

Merkel, et al., Statistical Detection of Malicious Pe-Executables for Fast Offline Analysis, IFIP International Conference on Communications and Multimedia Security, Springer, Berlin, 2010, 12 pages.

Singla, et al., "A Novel Approach to Malware Detection Using Static Classification," International Journal of Computer Science and Information Security 13.3, 2015, 5 pages.

Tabish, et al., "Malware Detection Using Statistical Analysis of Byte-Level File Content," Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics, ACM, 2009, 9 pages.

Treadwell, et al., A Heuristic Approach for Detection of Obfuscated Malware, 2009 IEEE International Conference on Intelligence and Security Informatics, 2009, 9 pages.

Notice of Allowance issued for U.S. Appl. No. 14/709,875, dated Nov. 14, 2019, 11 pages.

Corrected Notice of Allowance issued for U.S. Appl. No. 14/709,875, dated Dec. 23, 2019, 8 pages.

Corrected Notice of Allowance issued for U.S. Appl. No. 14/709,875, dated Jan. 24, 2020, 8 pages.

International Patent Application No. PCT/US2016/032090, International Search Report and Written Opinion dated Jul. 29, 2016, 13 pages.

Perdisci et al., "Classification of Packed Executables for Accurate Computer Virus Detection," Pattern Recognition Letters 29, 2008, 6 pages.

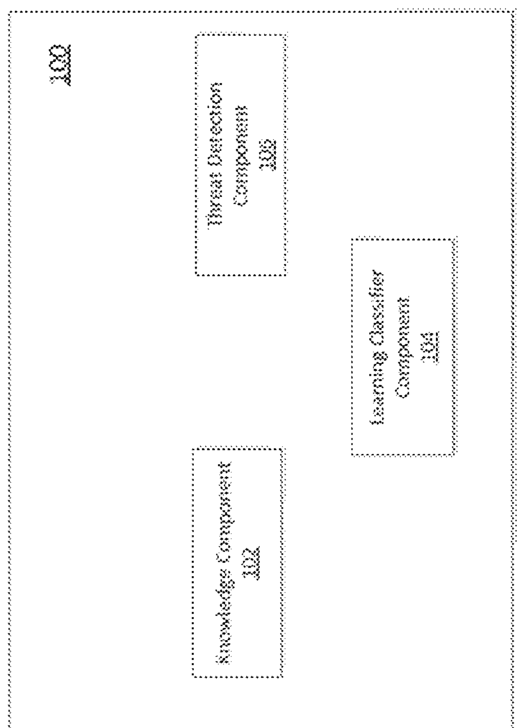
Perdisci et al., "McBoost: Boosting Scalability in Malware Collection and Analysis Using Statistical Classification of Executables," Annual Computer Security Applications, 2008, 10 pages.

Khan, et al., "Determining Malicious Executable Distinguishing Attributes and Low-Complexity Detection", J. ComputVirol, 7.2 (2011), 11 pgs.

Kolter, et al., "Leaming to Detect and Classift Malicious Executables in the Wild", Journal of Machine Learning Research 7 (2006), 24 pages.

* cited by examiner

FIG. 1


100
Threat Detection Component 106
Learning Classifier Component 104
Knowledge Component 102

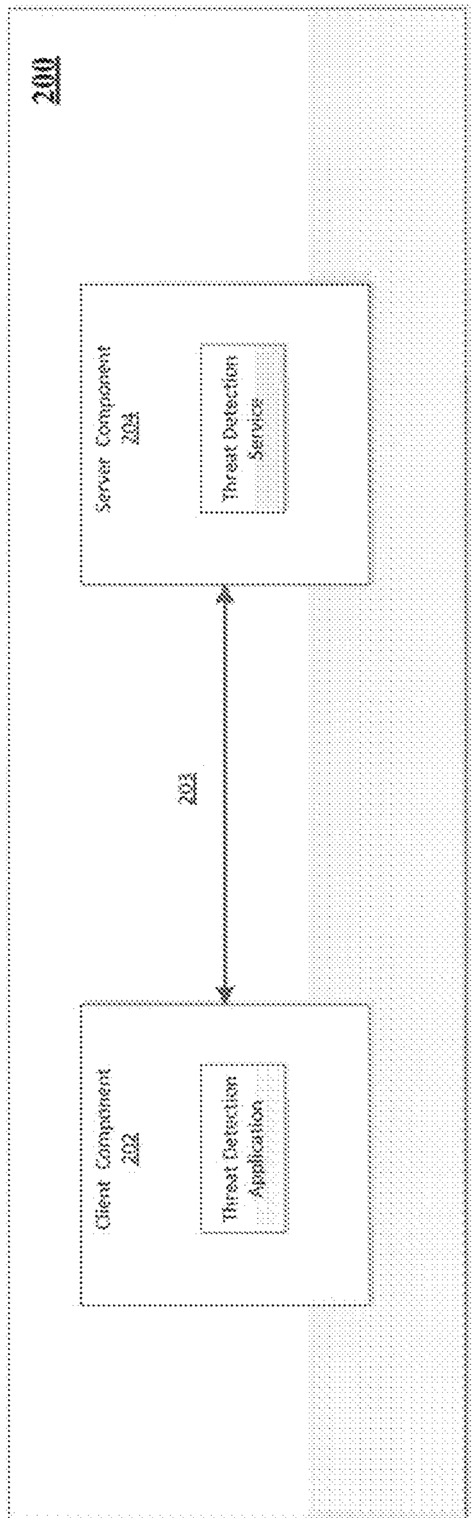
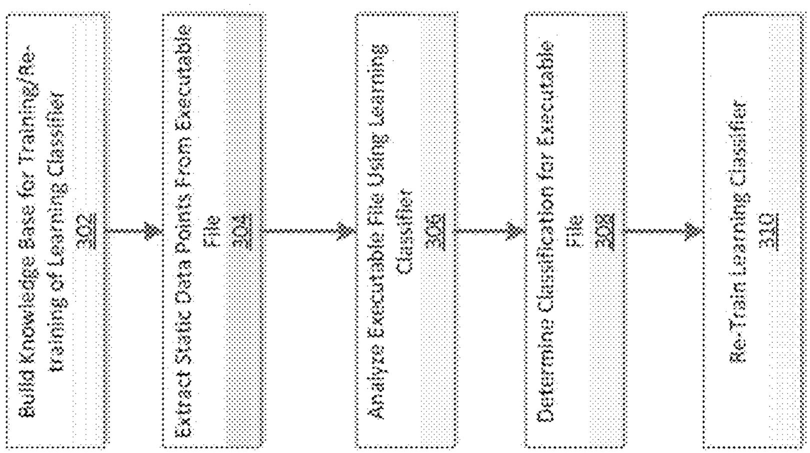

200
Server Component
204
Threat Detection Service
203
Client Component
202
Threat Detection Application

FIG. 2

FIG. 3

300


Build Knowledge Base for Training/Re-training of Learning Classifier
302
Extract Static Data Points From Executable File
304
Analyze Executable File Using Learning Classifier
306
Determine Classification for Executable File
308
Re-Train Learning Classifier
310

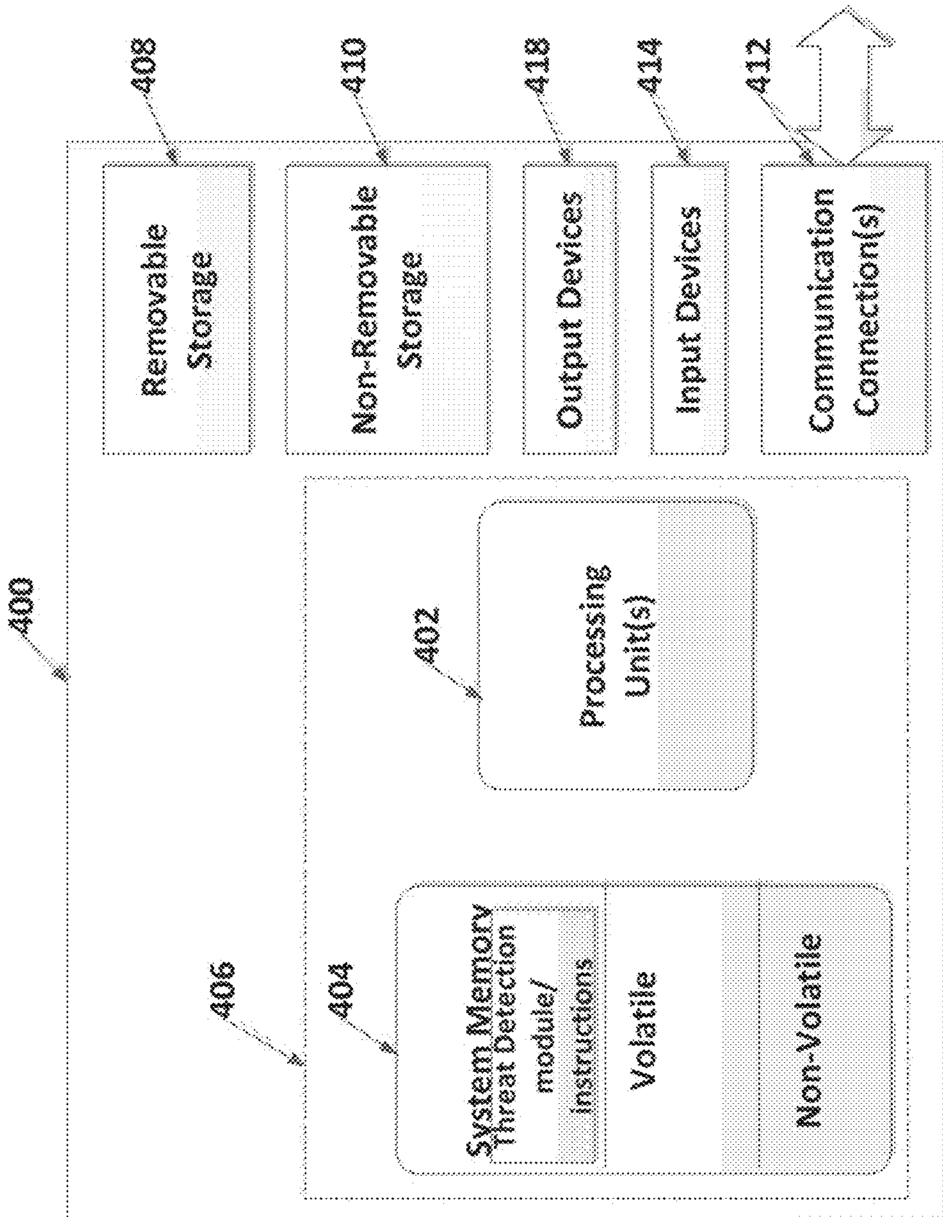

400
408
410
418
414
412
Removable Storage
Non-Removable Storage
Output Devices
Input Devices
Communication Connection(s)
402
Processing Unit(s)
406
404
System Memory
Threat Detection module/ instructions
Volatile
Non-Volatile

FIG. 4

AUTOMATIC THREAT DETECTION OF EXECUTABLE FILES BASED ON STATIC DATA ANALYSIS

CROSS-REFERENCE TO RELATED APPLICATION(S)

This application is a continuation of, and claims a benefit of priority from U.S. patent application Ser. No. 14/709,875, filed May 12, 2015, issued as U.S. Pat. No. 10,599,844, entitled "AUTOMATIC THREAT DETECTION OF EXECUTABLE FILES BASED ON STATIC DATA ANALYSIS," which is fully incorporated by reference herein.

BACKGROUND

Everyday new executable files are created and distributed across networks. A large portion of these distributed executable files are unknown. For instance, it is not known if such distributed executable files are malicious or not. Given the high volume of new unknown files distributed on a daily basis, it is important to determine threats contained in the set of new unknown files instantaneously and accurately. It is with respect to this general environment that aspects of the present technology disclosed herein have been contemplated.

SUMMARY

Aspects of the present disclosure relate to threat detection of executable files. A plurality of static data points are extracted from an executable file without decrypting or unpacking the executable file. The executable file may then be analyzed without decrypting or unpacking the executable file. Analyzing of the executable file comprises applying a classifier to the plurality of static data points extracted from the executable file. The classifier is trained from data comprising known malicious executable files, known benign executable files and potentially unwanted executable files. Based upon the analysis of the executable file, a determination is made as to whether the executable file is harmful. In some examples, execution of the executable file is prevented when a determined probability value that the executable file is harmful exceeds a threshold value.

This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used to limit the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

Non-limiting and non-exhaustive examples are described with reference to the following figures. As a note, the same number represents the same element or same type of element in all drawings.

FIG. **1** illustrates an exemplary system **100** showing interaction of components for implementation threat detection as described herein.

FIG. **2** illustrates an exemplary distributed system **200** showing interaction of components for implementation of exemplary threat detection as described herein.

FIG. **3** illustrates an exemplary method **300** for implementation of threat detection systems and methods described herein.

FIG. **4** illustrates one example of a suitable operating environment **400** in which one or more of the present examples may be implemented.

Non-limiting examples of the present disclosure relate to the threat detection of executable files. The examples disclosed herein may also be employed to detect zero day threats from unknown executable files. However, one skilled in the art will recognize that the present disclosure is not limited to detection of executable files that may present zero day threats and can be applicable to any unknown file that is attempting to execute on a system (e.g., processing device). The present disclosure is able to detect whether an executable file is harmful or benign before the executable file is actually executed on a processing device. In examples, machine learning processing applies a classifier to evaluate an executable file based on static points data collected from the executable file. The classifier is trained from a collection of data comprising known malicious files, potentially unwanted files and benign files. The classifier is designed and trained such that it can handle encrypted and/or compressed files without decrypting and/or decompressing the files.

Approaches to detecting threats typically focus on finding malicious code blocks within a file and analyzing the behavior of the file. Such approaches are expensive and time-consuming operations that require decrypting encrypted files, dissembling code, analyzing behavior of malware, among other things. Additionally, behavioral detection requires the execution of the potentially malicious code, thereby presenting an opportunity for the malicious code to harm the computing system is executing on. The present disclosure achieves high and accurate classification rates for potentially malicious executables without the need to employ time consuming processing steps like decryption, unpacking or executing unknown files while maintaining a controlled and safe environment. The determination of unknown files is achieved by evaluating static data points extracted from an executable file using a trained classification system that is able to identify potential threats without analyzing executable behavior of a file. The present disclosure also provides for the creation of a training set that adaptively learns what static data points may indicate the existence of malicious code using training data that contains a statistically significant number of both encrypted and non-encrypted examples of malicious or unwanted executable files, among other information. By collecting a large set of training examples as they appear as publicly available and distributed over network connection (e.g., "in the wild"), the present disclosure ensures that its adaptive learning processing is robust enough to comprise sufficient representation of features (and distributions) from files that may be encrypted, not-encrypted, compressed, and uncompressed, among other examples.

A number of technical advantages are achieved based on the present disclosure including, but not limited to: enhanced security protection including automatic detection of threats, reduction or minimization of error rates in identification and marking of suspicious behavior or files (e.g., cut down on the number of false positives), ability to adapt over time to continuously and quickly detect new threats or potentially unwanted files/applications, improved efficiency in detection of malicious files, and improved usability and interaction for users by eliminating the need to continuously check for security threats, among other benefits that will be apparent to one of skill in the art.

FIG. **1** illustrates an exemplary system **100** showing an interaction of components for implementation threat detec-

tion as described herein. Exemplary system **100** may be a combination of interdependent components that interact to form an integrated whole for execution of threat detection and/or prevention operations. Components of the systems may be hardware components or software implemented on and/or executed by hardware components of the systems. In examples, system **100** may include any of hardware components (e.g., used to execute/run operating system (OS)), and software components (e.g., applications, application programming interfaces, modules, virtual machines, runtime libraries, etc.) running on hardware. In one example, an exemplary system **100** may provide an environment for software components to run, obey constraints set for operating, and/or makes use of resources or facilities of the system **100**, where components may be software (e.g., application, program, module, etc.) running on one or more processing devices. For instance, threat detection operations (e.g., application, instructions, modules, etc.) may be run on a processing device such as a computer, a client device (e.g., mobile processing device, laptop, smartphone/phone, tablet, etc.) and/or any other electronic devices, where the components of the system may be executed on the processing device. In other examples, the components of systems disclosed herein may be spread across multiple devices. For instance, files to be evaluated may be present on a client device and information may be processed or accessed from other devices in a network such as, for example, one or more server devices that may be used to perform threat detection processing and/or evaluation a file before execution of the file by the client device.

As one example, system **100** comprises a knowledge component **102**, a learning classifier component **104**, and a threat determination component **106**, each having one or more additional components. The scale of systems such as system **100** may vary and include more or less components than those described in FIG. **1**. In alternative examples of system **100**, interfacing between components of the system **100** may occur remotely, for example where threat detection processing is implemented on a first device (e.g., server) that remotely monitors and controls process flow for threat detection and prevention of a second processing device (e.g., client).

As an example, threat detection may detection exploits that are executable files. However, one skilled in the art will recognize that the descriptions herein referring to executable files are just an example. Threat detection examples described herein can relate to any computer file. In one example, an executable file may be a portable executable (PE) file that is a file format for executables, object code, dynamic link library files (DLLs), and font library files, among other examples. However, one skilled in the art will recognize that executable files are not limited to PE files and can be any file or program that executes a task according to an encoded instruction. Knowledge component **102** described herein may collect data for use in building, training and/or re-training a learning classifier to evaluate executable files. The knowledge component **102** is one or more storages, memories, and/or modules that continuously collects and manages data that may be used to detect threats in files such as executable files. In one example, the knowledge component **102** maintains a robust collection of data comprising known malicious executable files, known benign executable files, and potentially unwanted executable files. As an example, Malicious executable files may be any type of code, data, objects, instructions, etc., that may cause harm or alter an intended function of a system and/or any system resources (e.g., memory, processes, peripherals, etc.) and/or applications running on a device such as an operating system (OS). A benign executable file is a file that upon execution, will not cause harm/damage or alter an intended system function. In other examples, a benign executable may cause harm/damage or alter an intended system; however, the potential harm/damage or alteration may be acceptable to the owner of the system or device. In examples, potentially unwanted executable files are files that may be installed on system **100** that may not be malicious, but a user of the device may not want such a file to execute and/or a file that is executed/installed unknowingly to a user. In one example, classification of executable files as malicious, benign or potentially unwanted are done by research and development support associated with developments and programming of a threat detection application/module. However, one skilled in the art will recognize that identification and classification of files into the above identified categories may be done by monitoring or evaluating a plurality of resources including but not limited to: network data, executable file libraries and information on previously known malicious executable files as well as benign executable files and potentially unwanted executable files, users/customers of threat detection/computer security software, network flow observed from use of threat detection processing and products, business associations (e.g., other existing threat detection services or partners), third-party feeds, and updates from threat detection performed using learning classifier of present disclosure, among other examples.

To classify executable files into one of the above identified categories, the knowledge component **102** collects static data on a large variety of executable files (e.g., PE files). Examples of different types of executable files collected and evaluated include but are not limited to: bit files (e.g., 32/64 bit files), operating system files (e.g., Windows, Apple, Linux, Unix, etc.), custom built files (e.g., internal tool files), corrupted files, partial downloaded files, packed files, encrypted files, obfuscated files, third party driver files, manually manipulated binary files, unicode files, infected files, and/or memory snapshots, among other examples. The data collected and maintained by the knowledge component **102** yields a knowledgebase that may be used to periodically train a classifier, e.g. learning classifier component **104** utilized by system **100**. The learning classifier component may be used to classify an executable file as one of the following classifications: malicious, benign or potentially unwanted. In some examples, classification may span two or more two or more of those classification categories. For example, an executable file may be benign in the sense that it is not harmful to system **100** but may also be classified as potentially unwanted as it might be installed without explicit user consent, for example. Data maintained by the knowledge component **102** may be continuously updated system **100** or a service that updates system **100** with new exploits to add to the training sample. For example, a research team may be employed to continuously collect new examples of harmful executables, benign executables, and potentially unwanted executables, as many unknown executable files are generated on a daily basis over the Internet. Executable files may be evaluated by the research team such as by applying applications or processing to evaluate executable files including data associated with the file and/or actions associated with the file (e.g., how it is installed and what a file does upon execution). Continuous update of the data maintained by the knowledge component **102** in conjunction with on-going re-learning/re-training of the learning classifier component **104** ensures that system **100** is up-to-date on current threats. Knowledge of the most current threats

improves the generalization capability of system **100** to new unknown threats. By incorporating malicious files, benign files and potentially unwanted files, the present disclosure greatly improves a knowledge base that may be used to in training a learning classifier thereby resulting in more accurate classifications as compared with other knowledge bases that are based on only malicious and/or benign files.

In examples, the collected data on the executable files is analyzed to identify static data points that may indicate one of a malicious file, a benign file or a potentially unwanted file. For instance, the knowledge component **102** may employ one or more programming operations to identify static data points for collection, and to associate the static data points with one of the categories of files (e.g., malicious, benign or potentially unwanted). Programming operations utilized by the knowledge component **102** include operations to collect file data (e.g., executable files or data points from executable files), parse the file data, and store extracted data points. In at least one example, the knowledge component **102** comprises one or more components to manage file data. For example, the knowledge component **102** may comprise one or more storages such as databases, and one or more additional components (e.g., processors executing programs, applications, application programming interfaces (APIs), etc.).

Further, the knowledge component **102** may be configured to continuously collect data, generate a more robust collection of file data to improve classification of file data and train a classifier used to detect whether an executable file is harmful, benign, or potentially unwanted. The identification of static data points to be collected and analyzed for executable files may be continuously updated as more information becomes available to the knowledge component **102**. A static data point may be a point of reference used to evaluate an executable file. As examples, static data points include, but are not limited to: header information, section information, import and export information, certificate information, resource information, string and flag information, legal information, comments, and/or program information (e.g., APIs and DLLs), among other examples. Static data points may be organized into categories that identify a type of static data point. Categories of static data points comprise, but are not limited to: numeric values, nominal values string sequences, byte sequences, and/or Boolean values, among other examples. Any number of static data points may be collected and analyzed for an executable file. In general, collecting and analyzing a greater number of static data points results in more accurate classification of an executable file. For example, eighty static data points may be identified and collected (or attempted to be collected) from an executable file during analysis of an executable file. While a specific number is static data points are provided herein, one of skill in the art will appreciate the more or fewer static data points may be collected without departing from the scope of this disclosure. As an example, the following table, Table 1.1, identifies some of the static data points identified for analysis of an executable file, where the static data points are organized by category:

TABLE 1.1

Numeric values	Nominal values	Strings/Byte sequences	Boolean values
File size	initialize	Comments	Address Of Entry Point Anomaly
linker version	Un-initialize	company name	Image Base Anomaly
code size	entry point	file description	Section Alignment Anomaly
OS version	subsystem	internal name	Size Of Code Mismatch Anomaly
image version	file subtype	legal copyright	Low Import Count Anomaly
subsystem version	language	original file	Entry Point Anomaly
file version number	file flags masks	private build	certificate Validity
product version number	file flags	product name	Certificate Exception
size of heapr	file OS	special build	Code Characteristics Anomaly
size of stackr	file type	product version	Code Name Anomaly
size of image	machine type	file version	Count Anomaly
PE header time	PE type	package code	Data Characteristics Anomaly
Section Entropy	section counts	product code	Data Name Anomaly
Sections count	DLL count	export DLL name	Export Exception
	DLL functions	assembly version	Large Number of DLLs Anomaly
	data directory	Certificate Issuer	flag DLL Name Anomaly
	export count	Certificate Subject	Number of Functions Anomaly
	Earliest Data Byte	Imports	Function Name Anomaly
	resources	Exports	PE Header Anomaly
	resources language	Section Names	High Section Count Anomaly
	resource Encoding	Non-resource section strings	PE Magic Validity
	resource code page		Resource Exception
	resource size		VR Code Ratio Anomaly
	DLL characteristics		Import Exception

Examples of the present disclosure need not distinguish between encrypted files and non-encrypted files. The static data points may be extracted from files regardless of whether or not they are encrypted and/or compressed. Given that the training set contains a statistically significant number of encrypted and non-encrypted examples, as well as compressed and decompressed files, a learning classifier (e.g., learning classifier component **104**) may be trained to identify features in the extracted data to identify malicious files. Since a very large majority of the files "in the wild" use few tools to encrypt the files (e.g., encryption algorithm, packers, etc.), the distribution of the data found across large number of files is preserved after encryption, although the actual content of the data is transformed into something different. By collecting a large set of training examples as they appear "in the wild", a sufficient representation of features and distributions associated with the represented features exist in a training set that are both encrypted and not-encrypted. The present disclosure provides at least one benefit over other threat detection applications/programs by utilizing a very large and diverse training sample, thereby enabling a more intelligent and adaptive learning classifier that is able to achieve high detection rates for threats and low false positive rates, among other benefits.

In addition to collecting and managing information to train a learning classifier, the knowledge component **102** is used to evaluate an executable file and extract/collect static data points for evaluation of the file by the learning classifier before the file is executed. In one example, the system automatically identifies a new file (e.g., unknown file) for evaluation. As examples, identification of an executable file for evaluation may occur in examples such as: when a download of a file is requested, while a download is being performed, evaluation of streaming data, when a new file is detected as attempting to execute (e.g., potentially unwanted file), and before an unknown file or file containing executable code that was not previously checked attempts to execute, among other examples. In another example, a user of a device with which components of system **100** are operating may identify a file to be evaluated. The knowledge component **102** collects as many static data points as it can to evaluate the executable file using a learning classifier built by the learning classifier component **102**. In an exemplary extraction, the knowledge component **102** extracts each of the static data points identified in Table 1.1. The learning classifier component **102** intelligently builds a learning classifier to evaluate a file based on the extracted static data points of the file and the data managed by the knowledge component **102**.

The learning classifier component **104** is a component used to evaluate an executable file using the information provided by the knowledge component **102**. The learning classifier component **104** interfaces with the knowledge component **102** and a threat detection component **106** for the system **100** to evaluate an executable file as a possible threat. As an example, the learning classifier **104** applies programming operations or machine learning processing to evaluate static data points extracted from a file to be analyzed. In doing so, the learning classifier component **104** builds a learning classifier based on information from the knowledge component **102** including the extracted static data points of a file and the information used to train/re-train the learning classifier (e.g., static data on the robust collection of executable files including variety of file types of malicious executable files, benign executable files, and potentially unwanted executable files).

As an example, the learning classifier can adaptively set features of a learning classifier based on the static data points extracted for a file. For example, ranges of data can be identified for static data points that may enable the learning classifier to controllably select (e.g., turn on/off) features for evaluation by learning classifier. In one instance where a static data point extracted for evaluation is a file size (e.g., numeric value), the learning classifier may turn on/off features for evaluation based on whether a the file size of the file is within a certain range. In another example, the learning classifier may detect that the file does not contain "legal information" in the file. Legal information may be any identifying information indicating data that conveys rights to one or more parties including: timestamp data, licensing information, copyright information, indication of intellectual property protection, etc. In an example where the learning classifier detects the presence of legal information does not exist in the executable file, this may trigger the learning classifier to adaptively check for additional features that might be indicative of a threat or malicious file as well as turn off other features related to an evaluation of the "legal information." The learning classifier component **104** utilizes programming operations or machine-learning processing to train/re-train the learning classifier to uniquely evaluate any file/file type. One of skill in the art will appreciate different types of processing operations may be employed without departing from the spirit of this disclosure. For example, the learning classifier component **104** may utilize an artificial neural network (ANN), a decision tree, association rules, inductive logic, a support vector machine, clustering analysis, and Bayesian networks, among other examples.

The learning classifier component **104** processes and encodes collected/extracted data from a file to make the static data points suitable for processing operations. The processing and encoding executed by the learning classifier component **104** may vary depending on the identified categories and/or type (e.g., numeric values, nominal values, string/byte sequences, Boolean values, etc.) of static data points collected by the knowledge component **102**. As an example, string sequence data as well as byte sequence data may be parsed and processed as n-grams and/or n-gram word prediction (e.g., word-grams). For instance, for a given string all unigrams, bigrams and so forth up to a given length n are generated and the counts of the individual n-grams are determined. The resulting counts of the unique n-grams string and/or byte sequences are then used as input to a generated feature vector. In one example, strings are processed directly according to a bag of word model. A bag of word model is a simplifying representation used in natural language processing and information retrieval. In another example, numeric value static data points may be binned appropriately ensuring a good balance between information loss through data binning and available statistics. Nominal values as well Boolean values may be encoded using true/false flags. All of the different encoded data may be combined into one or more final feature vectors, for example after a complex vector coding/processing is performed (e.g., L2 norm).

As an application example of processing performed by the learning classifier component **104**, suppose extraction of static data points from a file identifies the following four (4) data points:

File size: 4982784

PEHeader Anomaly: False

Section name: .TEXT

Legal copyright: Copyright (C) Webroot Inc. 1997

Processing these 4 data points into a feature vector may require binning the file size, labeling the PE Header Anomaly, build n-grams for the section name and building word-gram for legal copyright. Encapsulation of special characters that sometimes appear in text (e.g., section names) may be desirable. In an example, strings of data may be transformed first into hex code representation and then build n-grams or word-grams. In this particular example, the section name is transformed into n-grams while the legal copyright text is transformed into word-grams. In evaluation using the classifier, the features related to the extracted data points may be weighted by the particular data point being evaluated or by category, for example. For instance, the training of the learning classifier may indicate that legal information (e.g., "legal copyright") provides a better indication that a file may be malicious than that of a section name. Features being evaluated (and their distributions) differ for benign files as compared to malicious files (e.g., harmful executables such as malware). For example, the data field "Legal Copyright" for benign files tends to have meaningful words, whereas in a malicious file, this data field is often left empty or it is filled with random characters. Commercial software files tend to have a valid certificate, whereas malware in general do not have valid certificates. Similarly, each data field for a static data point evaluated

provides further indication about the maliciousness of the file. The combined information from all these data fields coupled with machine learning enables accurate prediction determination as to whether a file is malicious, benign or potentially unwanted. As an example, the resulting feature vector in sparse form is shown below:

```
secname__002e00740065:0.204124145232
secname__0065:0.0944911182523
secname__0074006500780074:0.353553390593
secname__006500780074:0.408248290464
secname__00740065:0.144337567297
secname__002e007400650078:0.353553390593
secname__00650078:0.144337567297
secname__002e0074:0.144337567297
secname__002e:0.0944911182523
secname__00780074:0.433012701892
secname__0078:0.0944911182523
secname__007400650078:0.204124145232
secname__0074:0.472455591262
legalcopyright__0043006f0070007900720069006700680074:0.4472135955
legalcopyright__002800430029:0.4472135955
legalcopyright__0043006f00720070002e:0.4472135955
legalcopyright__0031003900390035:0.4472135955
legalcopyright__004d00690063007200 6f0073006f00660074:0.4472135955
filesize__9965:1.0
PEHeaderAnomaly__False:1.0
```

In examples, the learning classifier component **104** provides the one or more feature vectors as input to a support vector machine (SVM). The SVM may then perform data analysis and pattern recognition on the one or more feature vectors. In one example the SVM, may be a linear SVM. Given a set of training examples provided by the knowledge component **102**, an SVM may build a probabilistic model that indicates whether or not a file may be malicious. In another example, a hybrid approach may be elected that combines two or more individual linear SVM classifiers into a final classifier using ensemble methods. Specifically, the evaluated static data points may be subdivided into a set of families (e.g., sections, certificate, header data, and bytes sequences are used as different families). For each family, a linear SVM may be trained. The resulting classification scores generated by each linear SVM may then be combined into a final classification score using a decision tree. As an example, the decision tree may be trained using two-class logistic gradient boosting. In yet another example of feature classification evaluation, a classification may be subdivided into a three class classification problem defined by malicious files, potentially unwanted files/applications and benign files. The resulting three class problem is solved using multi-class classification (e.g., Directed Acyclic Graph (DAG) SVM) or, in case of the hybrid approach based on feature families, using a decision tree based on three-class logistic gradient boosting.

The threat detection component **106** is a component of system **100** that evaluates results of the processing performed by the learning classifier component **104** to make a determination as to whether the executable file is malicious, benign or a potentially unwanted file/application. As an example, a probabilistic value may be a final score determined from evaluating all static data points (or feature distributions) individually and determining an aggregate score that may be used for the evaluation of an executable file. In another example, correlation between static data points may be determined by the learning classification component **104** and a final score may be generated based on the correlation between static data points evaluated.

In one example, a classification as to whether a file is malicious or not may be based on comparison with a predetermined threshold value. For instance, a final score (e.g., probability value/evaluation) is determined based on feature vector processing performed by the learning classifier component **104** and compared with a probability threshold value for determining whether an executable file is malicious. As an example, a threshold value(s) may be set based on predetermined false positive range data where ranges may be set for one or more of the malicious executable files, the benign executable files and the potentially unwanted executable files. The threshold values for each of the different types may be the same or different. For instance, a threshold value may be set that indicates a confidence score in classifying an executable file as a malicious executable file. Ranges may be determined indicating how confident the system **100** is in predicting that a file is malicious. That may provide an indication of whether further evaluation of an executable file should occur. However, one skilled in the art will recognize that threshold determinations can be set in any way that be used to determine whether an executable file is malicious or not. Examples of further evaluation include but are not limited to: additional processing using a learning classifier, identification of an executable file as potentially malicious where services associated with the system **100** may follow-up, quarantining a file, and moving the file to a secure environment for execution (e.g., sandbox) among other examples. In other examples, retraining of the classifier may occur based on confidence scores. In examples, when a probability value exceeds or threshold value (or alternatively is less than a threshold value), it may be determined that an executable file may be identified as malicious. One skilled in the art will recognize that determination for predictive classification of executable files is not limited to threshold determinations. For example, any type of analytical, statistical or graphical analysis may be performed on data to classify an executable file/unknown executable file. As identified above, the threat detection component **106** may interface with the learning classifier component **104** to make a final determination as to how to classify an executable file as well as interface with the knowledge component **102** for training/retraining associated with the learning classifier.

FIG. **2** illustrates an exemplary distributed system **200** showing interaction of components for implementation of exemplary threat detection system as described herein. Where FIG. **1** illustrates an example system **100** having

components (hardware or software) operating on a client, FIG. **2** illustrates an exemplary distributed system **200** comprising a client component **202** connected with at least one server component **204** via communication line **203**. Communication line **203** represents the ability of the client component **202** to communicate with the server component **204**, for example, to send or receive information over a network connection. That is, client component **202** and server component **204** are connectable over a network connection (e.g., a connection Internet via, for example, a wireless connection, a mobile connection, a hotspot, a broadband connection, a dial-up, a digital subscriber line, a satellite connection, an integrated services digital network, etc.).

The client component **202** may be any hardware (e.g., processing device) or software (e.g., application/service or remote connection running on a processing device) that accesses a service made available by the server component **204**. The server component **204** may be any hardware or software (e.g., application or service running on a processing device) capable of communicating with the client component **202** for execution of threat detection processing (e.g., threat detection application/service). Threat detection processing may be used to evaluate a file before execution of the file as described in FIG. **1**. Threat detection application or services may be present on at least one of the client component **202** and the server component **204**. In other examples, applications or components (e.g., hardware or software) may be present on both the client component **202** and the server component **204** to enable processing by threat detection application/services when a network connection to the server cannot be established. In one example of system **200**, client component **202** (or server component **204**) may comprise one or more components for threat detection as described in system **100** including a knowledge component **102**, a learning classifier component **104** and/or a threat detection component **106**, as described in the description of FIG. **1**. In other examples, the client component **202** may transmit data to/from the server component **204** to enable threat detections services over distributed network **200**, for example as represented by communication line **203**. In one example, threat detection applications/services operating on the client component **202** may receive updates from the server component **204**. For instance, updates may be received by the client component **202** for re-training of a learning classifier used to evaluate an executable file on the client component **202**.

FIG. **3** illustrates an exemplary method **300** for performing threat detection. As an example, method **300** may be executed by an exemplary system such as system **100** of FIG. **1** and system **200** of FIG. **2**. In other examples, method **300** may be executed on a device comprising at least one processor configured to store and execute operations, programs or instructions. However, method **300** is not limited to such examples. Method **300** may be performed by any application or service that may include implementation of threat detection processing as described herein. The method **300** may be implemented using software, hardware or a combination of software and hardware.

Method **300** begins at operation **302** where a knowledge base is built for training/retraining of a learning classifier used to detect threats in executable files. Operation **302** builds the knowledge base from data collected and evaluated related to known malicious executable files, known benign executable files and potentially unwanted executable files as described in the description of FIG. **1** (e.g., knowledge component **102**). The knowledge base may be used to automatically train/re-train one or more learning classifiers used to evaluate threats in executable files based on the known malicious executable files, known benign executable files and the potentially unwanted executable files. In examples, the knowledge base may be maintained on at least one of a client component and a server component, and the knowledge base may be continuously updated with information from the resources described in FIG. **1** including update information based on threat detection processing evaluation performed on unknown executable files.

When an executable file is identified for evaluation, flow proceeds to operation **304** where a one or more static data points are extracted from an executable file. In operation **304**, an executable file is analyzed using machine learning processing as described with respect to the knowledge component **102** of FIG. **1** to collect/extract static data points from an executable file for evaluation. In examples, operation **304** occurs without decrypting or unpacking the executable file. However, in other examples, machine learning processing performed has the capability to evaluate static data points from extracted/unpacked content. In one example, machine learning processing is used to extract static data points from encrypted and/or compress versions of one or more files. In another example, machine learning processing is used to extract static data points from decrypted and/or unpacked versions of one or more files. In any example, extraction of static data points from different files (e.g. executable files) can be used to enhance training of a learning classifier, providing better results for extraction of static data points and classification of files.

In examples, operation **304** further comprises classifying extracted data according to a type of data extracted. For instance, the actions performed at operation **304** may be used to classify the plurality of static data points extracted into categories (categorical values) comprising numeric values, nominal values, string or byte sequences, and Boolean values, for example, as described with respect to FIG. **1**. In examples, data of an executable file (e.g., binary file data) may be parsed and compared against data maintained by a threat detection application/service as described in the present disclosure to determine static data points for extraction/collection.

In operation **306**, the executable file is analyzed for threats. As an example, operation **306** analyzes an executable without decrypting or unpacking the executable file. However, in other examples, the machine learning processing being performed has the capability to evaluate static data points from extracted/unpacked content. Operation **306** comprises applying a learning classifier (e.g., the learning classifier generated during performance of operation **302**) to the plurality of static data points extracted from the file. As discussed, the learning classifier may be built from data comprising known malicious executable files, known benign executable files and known unwanted executable files, for example. In one example, operation **306** comprises generating at least one feature vector from the plurality of static data points extracted using the learning classifier trained by the knowledge base. In order to generate the feature vector for the learning classifier, data may be parsed and encoded for machine learning processing.

In one example, generation of the feature vector may comprise selectively setting features of the learning classifier based on the one or more of static data points extracted. Features of the generated feature vector may be weighted based on classified categories identified by the knowledge base (as described above) and the plurality of static data points extracted from the file. As an example, one or more

features of the feature vector may be selectively turned on or off based on evaluation of whether a value of a static data point is within a predetermined range. However, one skilled in the art will recognize that the learning classifier can uniquely generate a feature vector for analysis of the executable file based on any data used to train/re-train the learning classifier. In examples, operation **306** further comprises evaluating the feature vector using linear or nonlinear support vector processing to determine a classification for the executable file, for example whether the executable file is harmful, benign, or unwanted.

Flow proceeds to operation **308**, where a determination is made as to a classification of the executable file. For example, operation **308** makes a final determination as to whether the executable file is harmful (e.g., malicious, malware) or not based on results of the analysis of the executable file (e.g., using machine learning processing by a learning classifier). In one example, results of the analysis of an executable file may be data obtained from learning classifier, such as, for example an SVM, processing data. In one example, operation **308** further comprises preventing execution of the executable file when a probability value that the executable file is harmful exceeds a threshold value. The probability value for the executable file may be determined based on applying the learning classifier to the executable file. As an example, the threshold value may be set based on predetermined false positive range data for identifying a malicious or harmful executable file. False positive range data may be determined from the analysis/evaluation of the known malicious executable files, known benign files and potentially unwanted executable files/applications, of the knowledge base. However, as acknowledged above, determining a classification of an executable file may be based on any type of analytical, statistical or graphical analysis, or machine learning processing. In one example, ranges can be based on evaluation of data during operation of the threat detection service as well as analysis related to unknown files, for example analytics performed to evaluate unknown files.

At any point in time, operation **310** may occur where a learning classifier used for threat detection processing is re-trained. Continuously re-training of the learning classifier may ensure that the threat detection application/service is up to date and able to accurately detect new threats. As identified above, re-training may occur through results of threat detection processing including updated information added to the knowledge base. In one example, training of a learning classifier can be based on evaluation of data during operation of the threat detection service as well as analysis related to unknown files, for example analytics performed to evaluate unknown files.

FIG. **4** and the additional discussion in the present specification are intended to provide a brief general description of a suitable computing environment in which the present invention and/or portions thereof may be implemented. Although not required, the embodiments described herein may be implemented as computer-executable instructions, such as by program modules, being executed by a computer, such as a client workstation or a server. Generally, program modules include routines, programs, objects, components, data structures and the like that perform particular tasks or implement particular abstract data types. Moreover, it should be appreciated that the invention and/or portions thereof may be practiced with other computer system configurations, including hand-held devices, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

FIG. **4** illustrates one example of a suitable operating environment **400** in which one or more of the present embodiments may be implemented. This is only one example of a suitable operating environment and is not intended to suggest any limitation as to the scope of use or functionality. Other well-known computing systems, environments, and/or configurations that may be suitable for use include, but are not limited to, personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, programmable consumer electronics such as smart phones, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

In its most basic configuration, operating environment **400** typically includes at least one processing unit **402** and memory **404**. Depending on the exact configuration and type of computing device, memory **404** (storing, among other things, executable evaluation module(s), e.g., malware detection applications, APIs, programs etc. and/or other components or instructions to implement or perform the system and methods disclosed herein, etc.) may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.), or some combination of the two. This most basic configuration is illustrated in FIG. **4** by dashed line **406**. Further, environment **400** may also include storage devices (removable, **408**, and/or non-removable, **410**) including, but not limited to, magnetic or optical disks or tape. Similarly, environment **400** may also have input device(s) **414** such as keyboard, mouse, pen, voice input, etc. and/or output device(s) **416** such as a display, speakers, printer, etc. Also included in the environment may be one or more communication connections, **412**, such as LAN, WAN, point to point, etc.

Operating environment **400** typically includes at least some form of computer readable media. Computer readable media can be any available media that can be accessed by processing unit **402** or other devices comprising the operating environment. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media includes volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other non-transitory medium which can be used to store the desired information. Computer storage media does not include communication media.

Communication media embodies computer readable instructions, data structures, program modules, or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes

wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of the any of the above should also be included within the scope of computer readable media.

The operating environment **400** may be a single computer operating in a networked environment using logical connections to one or more remote computers. The remote computer may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above as well as others not so mentioned. The logical connections may include any method supported by available communications media. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

The different aspects described herein may be employed using software, hardware, or a combination of software and hardware to implement and perform the systems and methods disclosed herein. Although specific devices have been recited throughout the disclosure as performing specific functions, one of skill in the art will appreciate that these devices are provided for illustrative purposes, and other devices may be employed to perform the functionality disclosed herein without departing from the scope of the disclosure.

As stated above, a number of program modules and data files may be stored in the system memory **404**. While executing on the processing unit **402**, program modules **408** (e.g., applications, Input/Output (I/O) management, and other utilities) may perform processes including, but not limited to, one or more of the stages of the operational methods described herein such as method **300** illustrated in FIG. **3**, for example.

Furthermore, examples of the invention may be practiced in an electrical circuit comprising discrete electronic elements, packaged or integrated electronic chips containing logic gates, a circuit utilizing a microprocessor, or on a single chip containing electronic elements or microprocessors. For example, examples of the invention may be practiced via a system-on-a-chip (SOC) where each or many of the components illustrated in FIG. **4** may be integrated onto a single integrated circuit. Such an SOC device may include one or more processing units, graphics units, communications units, system virtualization units and various application functionality all of which are integrated (or "burned") onto the chip substrate as a single integrated circuit. When operating via an SOC, the functionality described herein may be operated via application-specific logic integrated with other components of the operating environment **400** on the single integrated circuit (chip). Examples of the present disclosure may also be practiced using other technologies capable of performing logical operations such as, for example, AND, OR, and NOT, including but not limited to mechanical, optical, fluidic, and quantum technologies. In addition, examples of the invention may be practiced within a general purpose computer or in any other circuits or systems.

This disclosure described some aspects of the present technology with reference to the accompanying drawings, in which only some of the possible embodiments were shown. Other aspects may, however, be embodied in many different forms and should not be construed as limited to the embodiments set forth herein. Rather, these aspects were provided so that this disclosure was thorough and complete and fully conveyed the scope of the possible embodiments to those skilled in the art.

Although specific aspects were described herein, the scope of the technology is not limited to those specific embodiments. One skilled in the art will recognize other embodiments or improvements that are within the scope and spirit of the present technology. Therefore, the specific structure, acts, or media are disclosed only as illustrative embodiments. The scope of the technology is defined by the following claims and any equivalents therein.

What is claimed is:

1. A computer-implemented method comprising:

identifying, by a knowledge module, static data points that may be indicative of either a harmful or benign executable file;

associating, by the knowledge module, the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;

identifying an executable file to be evaluated;

extracting, by the knowledge module, a plurality of static data points from the identified executable file;

generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range; and

providing the generated feature vector to one or more support vector machines to build a probabilistic model that indicates whether the executable file fits into one of the categories of files.

2. The computer-implemented method according to claim **1**, wherein the plurality of static data points are extracted without decrypting or unpacking the executable file.

3. The computer-implemented method according to claim **1**, wherein the one or more support vector machines builds the probabilistic model by performing data analysis and pattern recognition on the one or more feature vectors.

4. The computer-implemented method according to claim **1**, wherein the probabilistic model indicates whether the executable file is harmful.

5. The computer-implemented method according to claim **1**, wherein the executable file is identified in response to a detected condition.

6. The computer-implemented method according to claim **5**, wherein the detected condition is user request for a file download.

7. The computer-implemented method according to claim **5**, wherein the detected condition is the detection of a new file attempting to execute.

8. The computer-implemented method according to claim **1**, wherein the plurality of static data points represent predefined character strings in the executable file.

9. The computer-implemented method according to claim **1**, wherein a determination of whether the executable file is harmful is used to retrain the classifier.

10. A system comprising:

at least one memory; and

at least one processor operatively connected with the memory and configured to perform operation of:

identifying static data points that may be indicative of either a harmful or benign executable file;

associating the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;

identifying an executable file to be evaluated;

extracting a plurality of static data points from the identified executable file; and

generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range; and

providing the generated feature vector to one or more support vector machines to build a probabilistic model that indicates whether the executable file fits into one of the categories of files.

11. The system according to claim **10**, wherein the plurality of static data points are extracted without decrypting or unpacking the executable file.

12. The system according to claim **10**, wherein the one or more support vector machines builds the probabilistic model by performing data analysis and pattern recognition on the one or more feature vectors.

13. The system according to claim **10**, wherein the probabilistic model indicates whether the executable file is harmful.

14. The system according to claim **10**, wherein the plurality of static data points represent predefined character strings in the executable file.

15. A computer-readable storage device containing instructions, that when executed on at least one processor, causing the processor to execute a process comprising:

identifying static data points that may be indicative of either a harmful or benign executable file;

associating the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;

identifying an executable file to be evaluated;

extracting a plurality of static data points from the identified executable file;

generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range; and

providing the generated feature vector to one or more support vector machines to build a probabilistic model that indicates whether the executable file fits into one of the categories of files.

16. The computer-readable storage device according to claim **15**, wherein the plurality of static data points are extracted without decrypting or unpacking the executable file.

17. The computer-readable storage device according to claim **15**, wherein the plurality of static data points represent predefined character strings in the executable file.

18. A computer-implemented method comprising:

identifying static data points that may be indicative of either a harmful or benign executable file;

associating the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;

identifying an executable file to be evaluated;

extracting a plurality of static data points from the executable file;

generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range; and

evaluating the feature vector using a machine learning model to determine whether the executable file fits into one of the categories of files.

19. The computer-implemented method according to claim **18**, wherein the plurality of static data points are extracted without decrypting or unpacking the executable file.

20. The computer-implemented method according to claim **18**, wherein the machine learning model comprises an artificial neural network.

21. The computer-implemented method according to claim **18**, wherein the machine learning model comprises a support vector machine.

22. The computer-implemented method according to claim **18**, wherein the machine learning model comprises a machine learning decision tree.

23. The computer-implemented method according to claim **18**, wherein the machine learning model comprises a Bayesian network.

24. The computer-implemented method according to claim **18**, wherein evaluating the feature vector using the machine learning model comprises evaluating the feature vector using support vector processing.

25. The computer-implemented method according to claim **18**, wherein the machine learning model comprises a clustering model.

26. The computer-implemented method according to claim **18**, wherein the executable file is identified in response to a detected condition.

27. The computer-implemented method according to claim **26**, wherein the detected condition is user request for a file download.

28. The computer-implemented method according to claim **26**, wherein the detected condition is the detection of a new file attempting to execute.

29. The computer-implemented method according to claim **18**, wherein the plurality of static data points represent predefined character strings in the executable file.

30. The computer-implemented method according to claim **18**, wherein a determination of whether the executable file is harmful is used to retrain the classifier.

31. The computer-implemented method according to claim **18**, further comprising preventing execution of the executable file based on classifying the executable file as harmful.

32. A computer-readable storage device containing instructions, that when executed on a processor, cause the processor to execute a process comprising:

identifying static data points that may be indicative of either a harmful or benign executable file;

associating the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;

identifying an executable file to be evaluated;

extracting a plurality of static data points from the executable file;

generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range; and

evaluating the feature vector using a machine learning model to determine whether the executable file fits into one of the categories of files.

33. The computer-readable storage device according to claim **32**, wherein the plurality of static data points are extracted without decrypting or unpacking the executable file.

34. The computer-readable storage device according to claim **32**, wherein the machine learning model comprises an artificial neural network.

35. The computer-readable storage device according to claim **32**, wherein the machine learning model comprises a support vector machine.

36. The computer-readable storage device according to claim **32**, wherein the machine learning model comprises a machine learning decision tree.

37. The computer-readable storage device according to claim **32**, wherein the machine learning model comprises a Bayesian network.

38. The computer-readable storage device according to claim **32**, wherein evaluating the feature vector using the machine learning model comprises evaluating the feature vector using support vector processing.

39. The computer-readable storage device according to claim **32**, wherein the machine learning model comprises a clustering model.

40. The computer-readable storage device according to claim **32**, wherein the executable file is identified in response to a detected condition.

41. The computer-readable storage device according to claim **40**, wherein the detected condition is user request for a file download.

42. The computer-readable storage device according to claim **40**, wherein the detected condition is the detection of a new file attempting to execute.

43. The computer-readable storage device according to claim **32**, wherein the plurality of static data points represent predefined character strings in the executable file.

44. The computer-readable storage device according to claim **32**, wherein a determination of whether the executable file is harmful is used to retrain the classifier.

45. The computer-readable storage device according to claim **32**, wherein the process further comprises preventing execution of the executable file based on classifying the executable file as harmful.

46. A system comprising:

at least one memory;

at least one processor configured to perform operation of:

identifying static data points that may be indicative of either a harmful or benign executable file; and

associating the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files; and

at least one processor configured to perform operation of:

identifying an executable file to be evaluated;

extracting a plurality of static data points from the executable file;

generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range; and

evaluating the feature vector using a machine learning model to determine whether the executable file fits into one of the categories of files.

47. The system according to claim **46**, wherein the plurality of static data points are extracted without decrypting or unpacking the executable file.

48. The system according to claim **46**, wherein the machine learning model comprises an artificial neural network.

49. The system according to claim **46**, wherein the machine learning model comprises a support vector machine.

50. The system according to claim **46**, wherein the machine learning model comprises a machine learning decision tree.

51. The system according to claim **46**, wherein the machine learning model comprises a Bayesian network.

52. The system according to claim **46**, wherein evaluating the feature vector using the machine learning model comprises evaluating the feature vector using support vector processing.

53. The system according to claim **46**, wherein the machine learning model comprises a clustering model.

54. The system according to claim **46**, wherein the executable file is identified in response to a detected condition.

55. The system according to claim **54**, wherein the detected condition is user request for a file download.

56. The system according to claim **54**, wherein the detected condition is the detection of a new file attempting to execute.

57. The system according to claim **46**, wherein the plurality of static data points represent predefined character strings in the executable file.

58. The system according to claim **46**, wherein a determination of whether the executable file is harmful is used to retrain the classifier.

59. The system according to claim **46**, further comprising at least one processor configured to perform operation of:

preventing execution of the executable file based on classifying the executable file as harmful.

* * * * *

Exhibit 9

US008181244B2

(12) **United States Patent**
Boney

(10) **Patent No.: US 8,181,244 B2**
(45) **Date of Patent: May 15, 2012**

(54) **BACKWARD RESEARCHING TIME STAMPED EVENTS TO FIND AN ORIGIN OF PESTWARE**

(75) Inventor: **Matthew L. Boney**, Longmont, CO (US)

(73) Assignee: **Webroot Inc.**, Broomfield, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 929 days.

(21) Appl. No.: **11/408,145**

(22) Filed: **Apr. 20, 2006**

(65) **Prior Publication Data**

US 2007/0250928 A1 Oct. 25, 2007

(51) **Int. Cl.**
G06F 21/00 (2006.01)

(52) **U.S. Cl.** **726/22**; 709/224; 713/100; 726/5; 726/23; 726/24; 726/27

(58) **Field of Classification Search** 726/22–25, 726/13; 713/187–188, 193–194
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,623,600 A 4/1997 Ji et al.
5,920,696 A 7/1999 Brandt et al.
5,951,698 A 9/1999 Chen et al.
6,069,628 A 5/2000 Farry et al.
6,073,241 A 6/2000 Rosenberg et al.
6,092,194 A 7/2000 Touboul
6,154,844 A 11/2000 Touboul
6,167,520 A 12/2000 Touboul
6,310,630 B1 10/2001 Kulkarni et al.
6,397,264 B1 5/2002 Stasnick et al.
6,405,316 B1 6/2002 Krishnan et al.
6,460,060 B1 10/2002 Maddalozzo, Jr. et al.
6,480,962 B1 11/2002 Touboul
6,535,931 B1 3/2003 Celi, Jr.
6,611,878 B2 8/2003 De Armas et al.
6,633,835 B1 * 10/2003 Moran et al. 702/190
6,667,751 B1 12/2003 Wynn et al.
6,701,441 B1 3/2004 Balasubramaniam et al.
6,772,345 B1 8/2004 Shetty
6,785,732 B1 8/2004 Bates et al.
6,804,780 B1 10/2004 Touboul
(Continued)

FOREIGN PATENT DOCUMENTS

WO WO 98/45778 10/1998
(Continued)

OTHER PUBLICATIONS

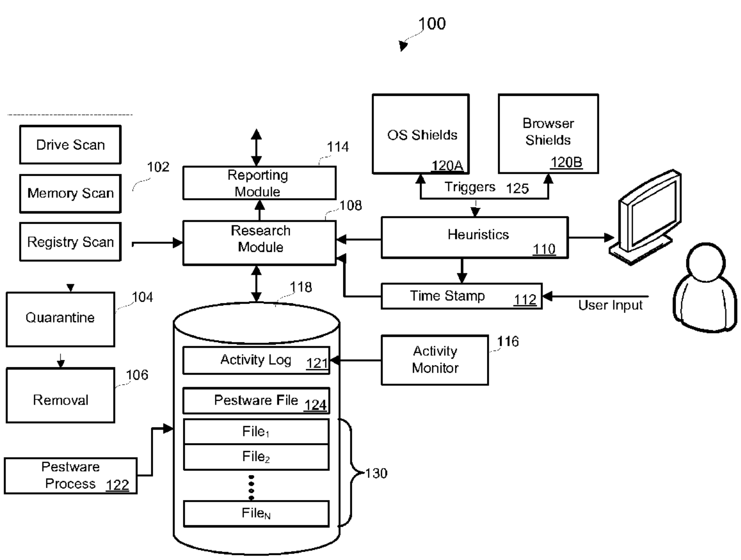
Elements of Computer Security|http://www.google.com/url?sa=t&rct=j&q=&esrc=s&frm=1&source=web&cd=8&ved=0CEwQFjAH&url=http%3A%2F%2Fcms.vle.ase.md%2Fget%2Fpdf%2F7664&ei=_pzyTp3mMeLL0QG7-9zLAg&usg=AFQjCNEmOz7OxykRPyP1_ePiXGgJI1WGWA&sig2=XE_fLI7CTuchuObMdbGYtA|David Salomon|2010.*
(Continued)

Primary Examiner — Taghi Arani
Assistant Examiner — Mahfuzur Rahman
(74) *Attorney, Agent, or Firm* — Cooley LLP

(57) **ABSTRACT**

A system and method for identifying an origin of suspected pestware activity on a computer is described. One embodiment includes establishing a time of interest relating to a suspicion of pestware on the computer, identifying, based upon the time of interest, indicia of pestware and accessing at least a portion of a recorded history of sources that the computer received files from so as to identify, based at least in part upon the identified indicia of pestware, a reference to an identity of a source that is suspected of originating pestware.

16 Claims, 5 Drawing Sheets



U.S. PATENT DOCUMENTS

6,813,711	B1	11/2004	Dimenstein	
6,829,654	B1	12/2004	Jungek	
6,910,134	B1	6/2005	Maher et al.	
6,965,968	B1	11/2005	Touboul	
7,058,822	B2	6/2006	Edery et al.	
7,107,617	B2	9/2006	Hursey et al.	
2002/0078381	A1	6/2002	Farley	
2002/0162015	A1	10/2002	Tang	
2002/0162017	A1	10/2002	Sorkin	
2002/0166063	A1	11/2002	Lachman et al.	
2003/0065943	A1*	4/2003	Geis et al.	713/201
2003/0074581	A1	4/2003	Hursey et al.	
2003/0101381	A1	5/2003	Mateev et al.	
2003/0159070	A1*	8/2003	Mayer et al.	713/201
2003/0217287	A1	11/2003	Kruglenko	
2004/0024864	A1*	2/2004	Porras et al.	709/224
2004/0030914	A1	2/2004	Kelley et al.	
2004/0034794	A1	2/2004	Mayer et al.	
2004/0064515	A1*	4/2004	Hockey	709/206
2004/0064736	A1	4/2004	Obrecht et al.	
2004/0080529	A1	4/2004	Wojcik	
2004/0143763	A1	7/2004	Radatti	
2004/0187023	A1	9/2004	Alagna et al.	
2004/0225877	A1*	11/2004	Huang	713/100
2005/0005160	A1	1/2005	Bates	
2005/0038697	A1	2/2005	Aaron	
2005/0138433	A1	6/2005	Linetsky	
2005/0154885	A1	7/2005	Viscomi et al.	
2005/0268112	A1*	12/2005	Wang et al.	713/188
2006/0075494	A1	4/2006	Bertman et al.	
2006/0075501	A1	4/2006	Thomas et al.	
2006/0136720	A1*	6/2006	Armstrong et al.	713/164
2006/0161988	A1	7/2006	Costea et al.	
2007/0094725	A1*	4/2007	Borders	726/22
2007/0168694	A1*	7/2007	Maddaloni et al.	714/4
2007/0220043	A1*	9/2007	Oliver et al.	707/103 R

FOREIGN PATENT DOCUMENTS

WO	PCT/US2007/067078	4/2007
WO	PCT/US2006/025378	9/2007
WO	PCT/US2007/067076	11/2007
WO	PCT/US2007/067084	11/2007

OTHER PUBLICATIONS

Codeguru, Three Ways to Inject Your Code Into Another Process, by Robert Kuster, Aug. 4, 2003, 22 pgs.

Codeguru, Managing Low-Level Keyboard Hooks With the Windows API for VB .Net, by Paul Kimmel, Apr. 18, 2004, 10 pgs.

Codeguru, Hooking the Keyboard, by Anoop Thomas, Dec. 13, 2001, 6 pgs.

Illusive Security, Wolves in Sheep's Clothing: malicious DLLs Injected Into trusted Host Applications, Author Unknown, http://home.arcor.de/scheinsicherheit/dll.htm 13 pgs.

DevX.com, Intercepting Systems API Calls, by Seung-Woo Kim, May 13, 2004, 6 pgs.

Microsoft.com, How to Subclass a Window in Windows 95, Article ID 125680, Jul. 11, 2005, 2 pgs.

MSDN, Win32 Hooks by Kyle Marsh, Jul. 29, 1993, 15 pgs.

PCT Search Report, PCT/US05/34874, Jul. 5, 2006, 7 Pages.

Yurcik, William et al., A Planning Framework for Implementing Virtual Private Networks, Jun. 2001, IT Pro, IEEE, pp. 41-44.

U.S. Appl. No. 10/956,573, filed Oct. 1, 2004, Steve Thomas.

U.S. Appl. No. 10/956,574, filed Oct. 1, 2004, Steve Thomas.

U.S. Appl. No. 10/956,818, filed Oct. 1, 2004, Matthew Boney.

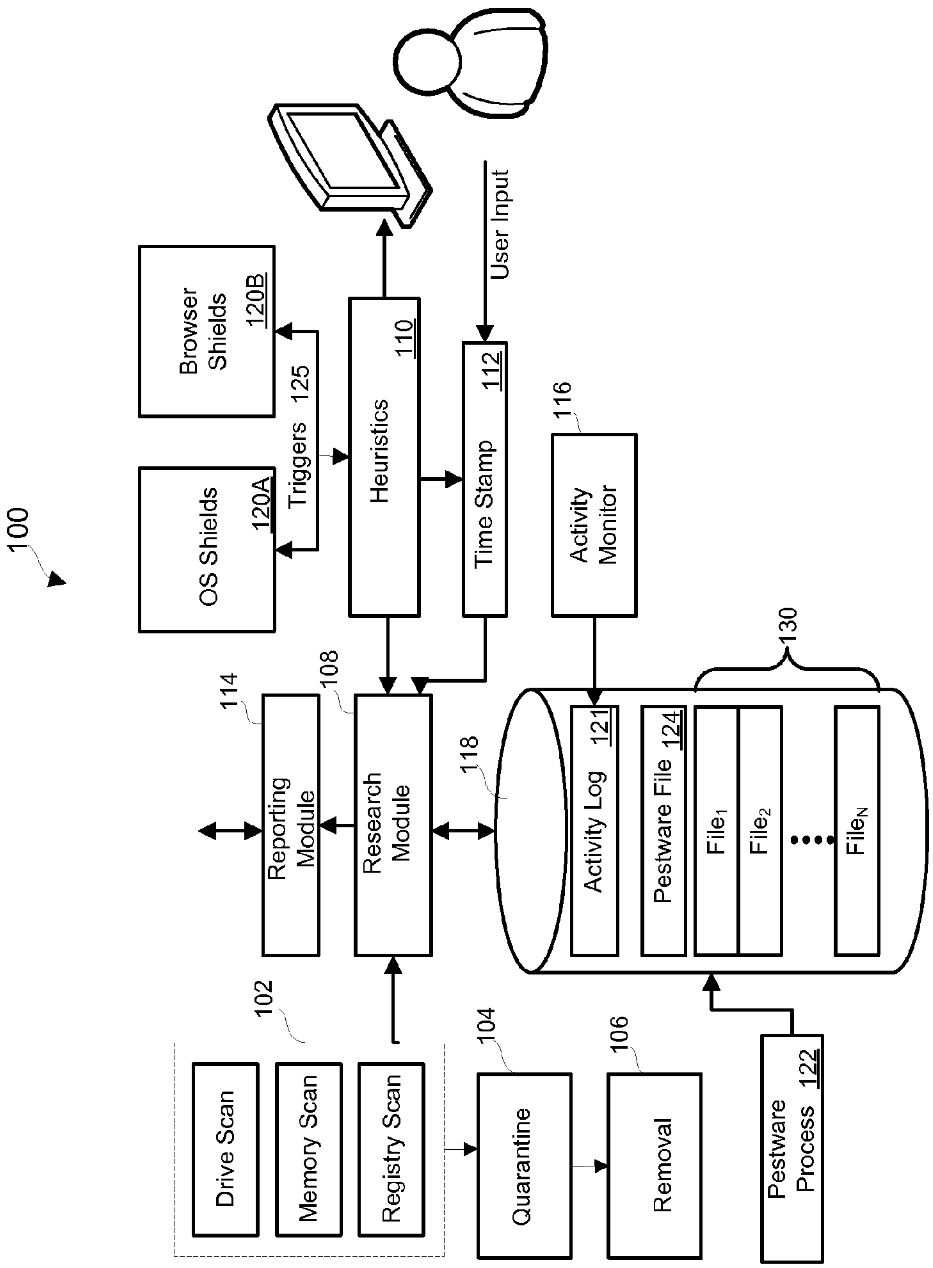
U.S. Appl. No. 10/257,609, filed Oct. 25, 2005, Michael Burtscher.

U.S. Appl. No. 11/237,291, filed Sep. 28, 2005, Philip Maddaloni.

U.S. Appl. No. 11/408,215, filed Apr. 20, 2006, Matthew Boney.

U.S. Appl. No. 11/408,146, filed Apr. 20, 2006, Matthew Boney.

* cited by examiner



100
OS Shields 120A
Browser Shields 120B
Triggers 125
Heuristics 110
User Input
Time Stamp 112
Activity Monitor
116
114
Reporting Module
108
Research Module
118
Activity Log 121
Pestware File 124
File1
File2
FileN
130
102
Drive Scan
Memory Scan
Registry Scan
104
Quarantine
106
Removal
Pestware Process 122

FIGURE 1

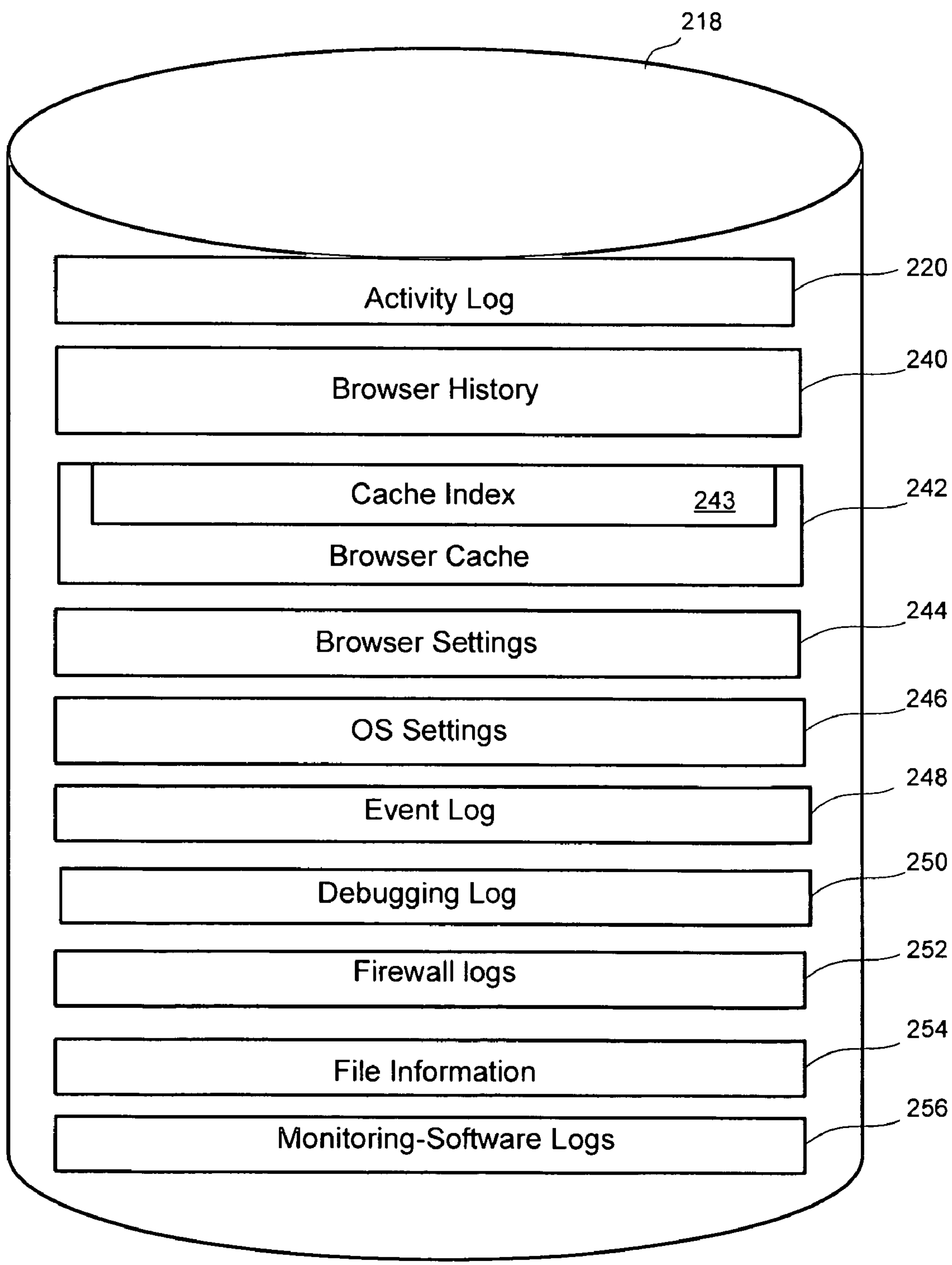

218
Activity Log
220
Browser History
240
Cache Index
243
Browser Cache
242
Browser Settings
244
OS Settings
246
Event Log
248
Debugging Log
250
Firewall logs
252
File Information
254
Monitoring-Software Logs
256

FIGURE 2

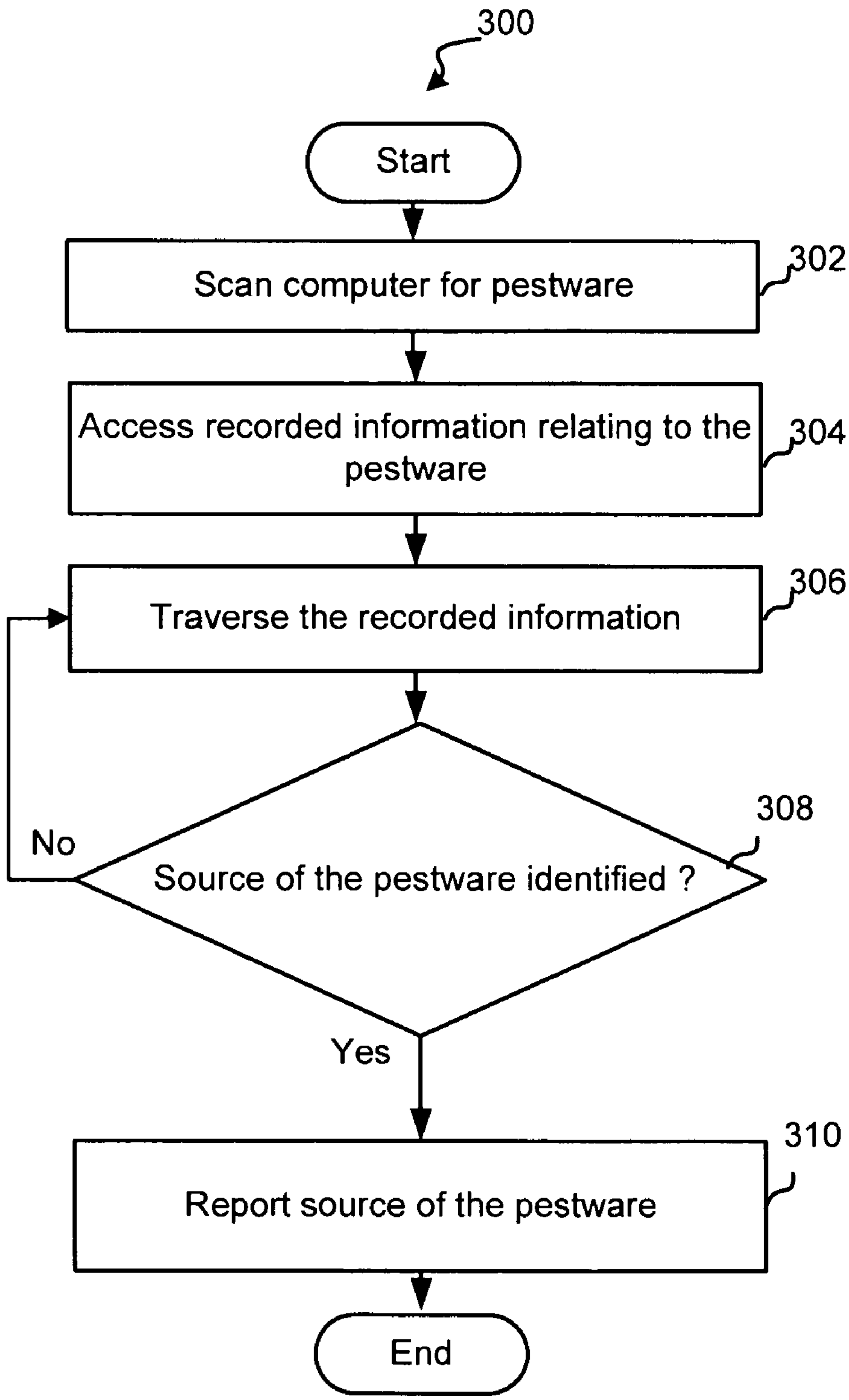

300
Start
Scan computer for pestware
302
Access recorded information relating to the pestware
304
Traverse the recorded information
306
No
Source of the pestware identified ?
308
Yes
Report source of the pestware
310
End

FIGURE 3

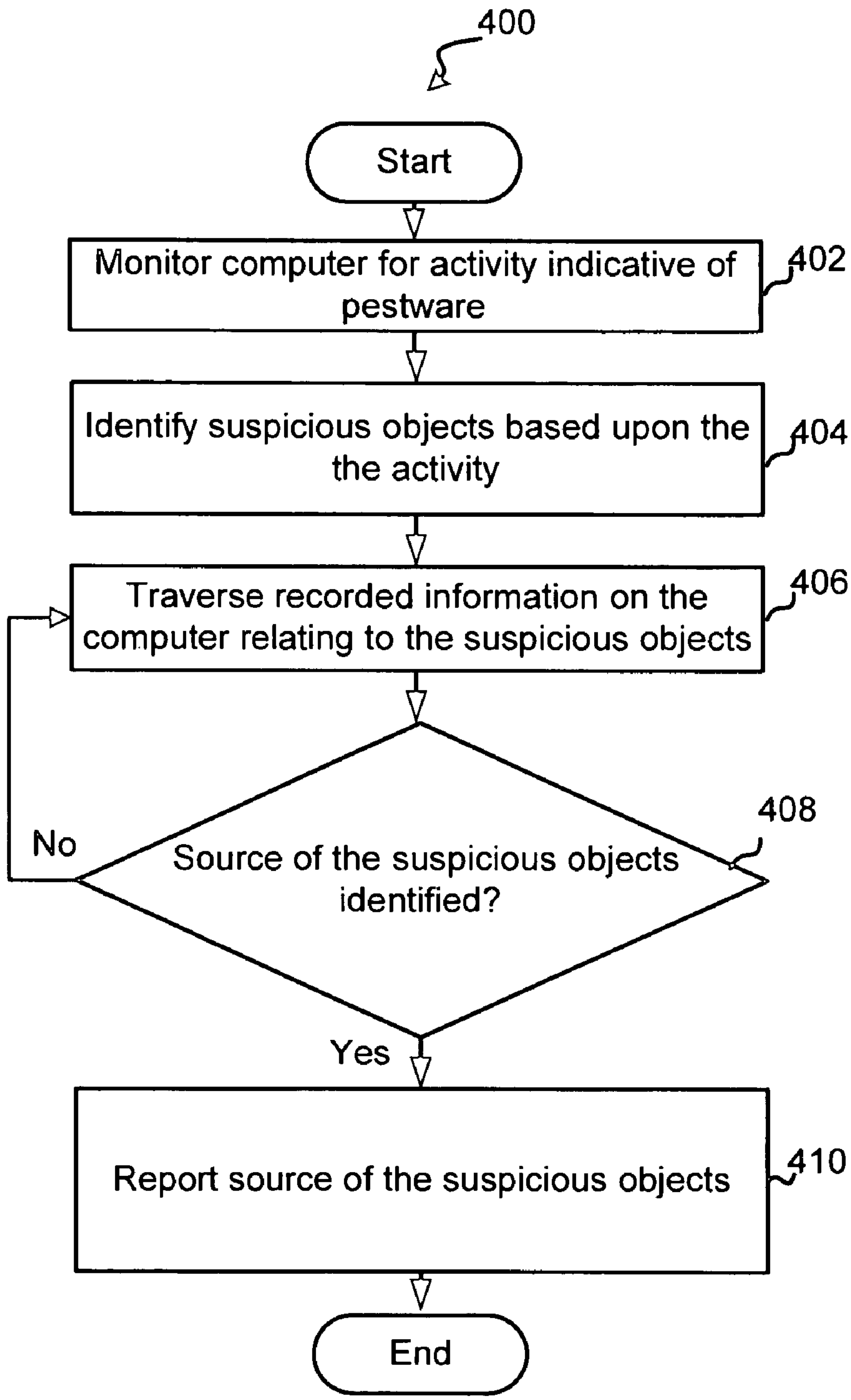

400
Start
Monitor computer for activity indicative of pestware
402
Identify suspicious objects based upon the the activity
404
Traverse recorded information on the computer relating to the suspicious objects
406
Source of the suspicious objects identified?
408
No
Yes
Report source of the suspicious objects
410
End

FIGURE 4

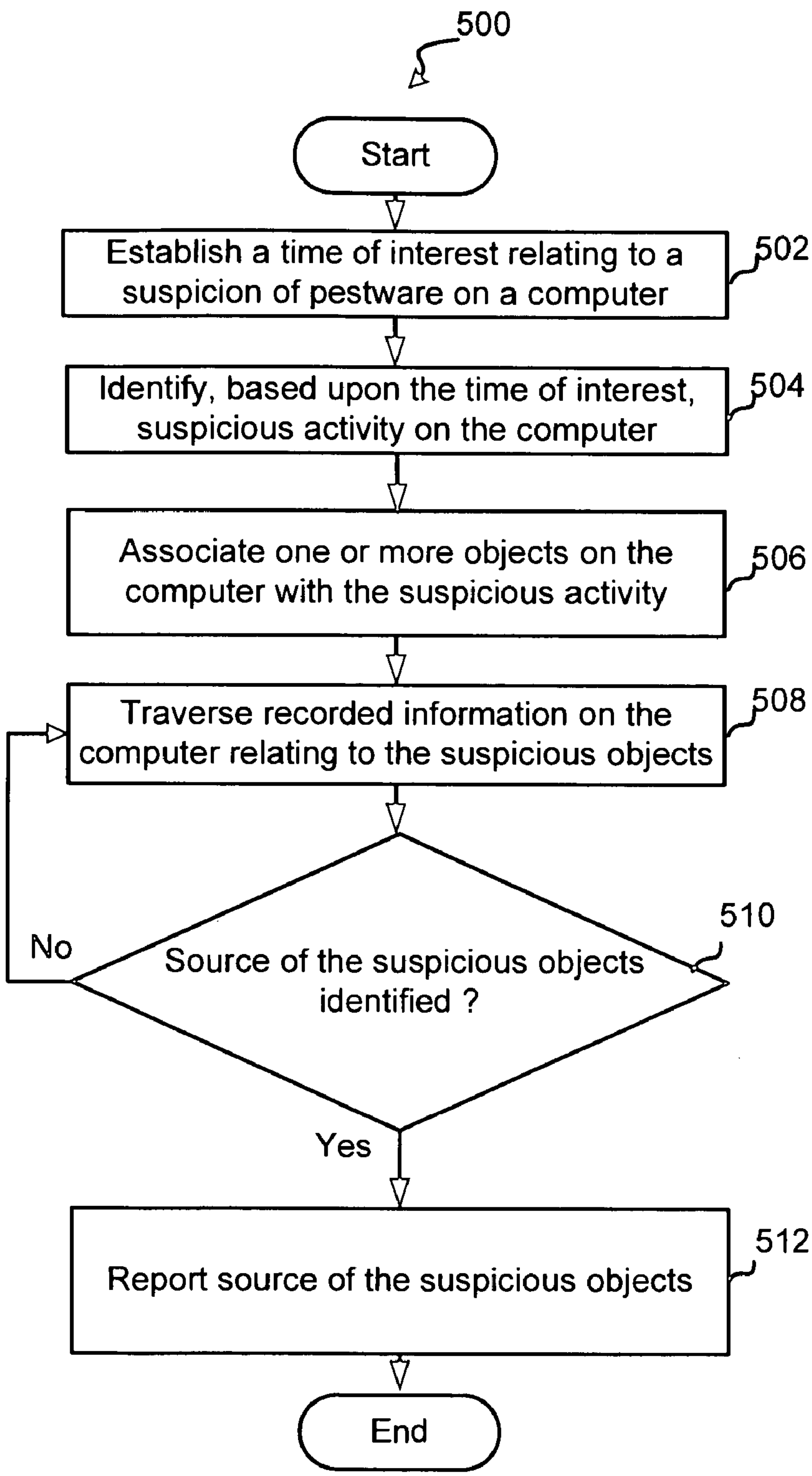

500
Start
Establish a time of interest relating to a suspicion of pestware on a computer
502
Identify, based upon the time of interest, suspicious activity on the computer
504
Associate one or more objects on the computer with the suspicious activity
506
Traverse recorded information on the computer relating to the suspicious objects
508
Source of the suspicious objects identified ?
510
No
Yes
Report source of the suspicious objects
512
End

FIGURE 5

BACKWARD RESEARCHING TIME STAMPED EVENTS TO FIND AN ORIGIN OF PESTWARE

RELATED APPLICATIONS

The present application is related to commonly owned and assigned application Ser. No. 10/956,573, now U.S. Pat. No. 7,480,683, entitled System and Method For Heuristic Analysis to Identify Pestware; application Ser. No. 10/956,574, now U.S. Pat. No. 7,533,131, entitled System and Method for Pestware Detection and Removal; application Ser. No. 10/956,818, entitled System and Method for Locating Malware and Generating Malware Definitions; application Ser. No. 11/257,609, entitled System and Method for Kernel-Level Pestware Management; application Ser. No. 11/237,291, entitled Client Side Exploit Tracking; application Ser. No. 11/408,146, entitled Backwards researching Activity Indicative of Pestware, filed herewith; and application Ser. No. 11/408,215, entitled Backward Researching Existing Pestware, filed herewith, which are incorporated herein by reference.

COPYRIGHT

A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

FIELD OF THE INVENTION

The present invention relates to computer system management. In particular, but not by way of limitation, the present invention relates to systems and methods for detecting, controlling and/or removing pestware.

BACKGROUND OF THE INVENTION

Personal computers and business computers are continually attacked by trojans, spyware, and adware, collectively referred to as "malware," "spyware" or "pestware." These types of programs generally act to gather information about a person or organization—often without the person or organization's knowledge. Some pestware is highly malicious. Other pestware is non-malicious but may cause issues with privacy or system performance. And yet other pestware is actually beneficial or wanted by the user. Unless specified otherwise, "pestware" as used herein refers to any of these programs that collects information about a person or an organization.

Software is presently available to detect and remove pestware. But as it evolves, the software to detect and remove it must also evolve. Accordingly, current techniques and software for removing pestware are not always satisfactory and will most certainly not be satisfactory in the future. Additionally, because some pestware is actually valuable to a user, pestware-detection software should, in some cases, be able to handle differences between wanted and unwanted pestware.

Current pestware removal software uses definitions of known pestware to search for and remove files on a protected system. These definitions are often slow and cumbersome to create. Additionally, it is often difficult to initially locate the pestware in order to create the definitions. Accordingly, a

system and method are needed to address the shortfalls of present technology and to provide other new and innovative features.

SUMMARY OF THE INVENTION

Exemplary embodiments of the present invention that are shown in the drawings are summarized below. These and other embodiments are more fully described in the Detailed Description section. It is to be understood, however, that there is no intention to limit the invention to the forms described in this Summary of the Invention or in the Detailed Description. One skilled in the art can recognize that there are numerous modifications, equivalents and alternative constructions that fall within the spirit and scope of the invention as expressed in the claims.

The present invention can provide a system and method for identifying an origin of suspected pestware activity on a computer. One embodiment includes establishing a time of interest relating to a suspicion of pestware on the computer, identifying, based upon the time of interest, indicia of pestware and accessing at least a portion of a recorded history of sources that the computer received files from so as to identify, based at least in part upon the identified indicia of pestware, a reference to an identity of a source that is suspected of originating pestware.

As previously stated, the above-described embodiments and implementations are for illustration purposes only. Numerous other embodiments, implementations, and details of the invention are easily recognized by those of skill in the art from the following descriptions and claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Various objects and advantages and a more complete understanding of the present invention are apparent and more readily appreciated by reference to the following Detailed Description and to the appended claims when taken in conjunction with the accompanying Drawings wherein:

FIG. **1** illustrates a block diagram of one implementation of the present invention;

FIG. **2** is a block diagram depicting an embodiment of the file storage device of FIG. **1**;

FIG. **3** is a flowchart depicting a method according to an exemplary embodiment;

FIG. **4** is a flowchart depicting another method according to another embodiment;

FIG. **5** is a flowchart depicting yet another method according to yet another embodiment.

DETAILED DESCRIPTION

Referring now to the drawings, where like or similar elements are designated with identical reference numerals throughout the several views, and referring in particular to FIG. **1**, it illustrates a block diagram of one implementation of the present invention. Shown is a protected computer **100** that includes a detection module **102**, quarantine module **104**, removal module **106**, and shields **120**. In addition, a research module **108** is shown coupled to a heuristics module **110**, a time stamp module **112** and a reporting module **114**.

Each of these modules can be implemented in software or hardware. And if implemented in software, the modules can be implemented in a single software package or in multiple software packages. In addition, one of ordinary skill in the art will recognize that the software can be designed to operate on any type of computer system including WINDOWS and

Linux-based systems. Additionally, the software can be configured to operate on personal computers and/or servers. For convenience, embodiments of the present invention are generally described herein with relation to WINDOWS-based systems. Those of skill in the art can easily adapt these implementations for other types of operating systems or computer systems.

Also shown is a file storage device **118** that is coupled to the research module **108** and an activity monitor **116**. In this embodiment the file storage device includes an activity log **121**, a pestware file **124** and a collection of N files **130**. The file storage device **118** is described herein in several implementations as hard disk drive for convenience, but this is certainly not required, and one of ordinary skill in the art will recognize that other storage media may be utilized without departing from the scope of the present invention. In addition, one of ordinary skill in the art will recognize that the storage device **118**, which is depicted for convenience as a single storage device, may be realized by multiple (e.g., distributed) storage devices.

In the exemplary embodiment, the pestware file **124**, corresponds to (e.g., includes data relating to) a pestware process **122** operating in memory. The pestware process **122** is exemplary of pestware processes that are configured to make one or more unauthorized alterations to the computer **100**. For example, the pestware process **122** may make changes to either or both of the browser settings and/or operating system (OS) settings without approval and/or the knowledge of the user.

In accordance with several embodiments, the research module **108** is configured to receive an indication that either known pestware is residing on the computer or activities indicative of pestware have occurred or are occurring on the protected computer. In response, the research module **108** is configured to research the activity log **121** and/or the N files **130**, which include information (e.g., historical logs) relating to events on the computer **100**, to identify a source of the pestware **122**, **124** or pestware-related activities, which may be reported by the reporting module **114** to a centralized data store for subsequent pestware management purposes.

For example, the identities (e.g., I.P. address, URL, email client or program name) of sources (e.g., web sites, email or program) of the pestware may be collected in a centralized data store (not shown) and then subsequently reported to other users. In addition, the sources of the pestware or suspected pestware may be visited to further research how the pestware is being distributed from the URLs and/or to generate new definitions for pestware discovered at these sources.

As described further with reference to FIG. **2**, the N files **130** include one or more files with information that assist the research module **108** in tracing information in the N files **130** to an identity (e.g., URL) of the source of pestware **122**, **124** on the computer **100**. One or more of the N files **130** may be associated with an operating system of the protected computer and/or one or more applications of the protected computer, and may include information such as process IDs, registry entries, file names, cookies and URLs among other information that is used to trace from an identified pestware file, pestware process and/or pestware activity to an originating source (e.g., URL or IP address) of the infection. In one embodiment, one or more of the N files **130** is generated from an application that generates a log of data after examining the computer. An example of such an application is an application distributed under the name HijackThis.

In the exemplary embodiment depicted in FIG. **1**, the research module **108** is configured to receive indications that pestware may be present on the computer from each of the detection engine **102**, the heuristics engine **110** and the time stamp module **112**, but this is certainly not required. In other embodiments, for example, the research module **108** may be configured to communicate only with the detection engine **102** or only with the heuristics engine. In yet other embodiments, the research module **108** may be configured to receive only a time stamp from the time stamp module **112**.

According to several embodiments, pestware-detection functions operating on the protected computer **100** are represented by the detection engine **102**, the quarantine engine **104**, the removal engine **106**, the shields **120** and the heuristic engine **110**. The basic functions of the detection engine **102** is to compare files, processes and registry entries on the protected computer against known pestware definitions and characteristics. When a match is found, in addition to quarantining and removing a pestware object, the detection engine **102** informs the research module **108** of the pestware. Details associated with several embodiments of sweep, quarantine, and removal engines are found in the above-identified application entitled System and Method for Pestware Detection and Removal.

Pestware and pestware activity can also be identified by the shields **120**, which generally run in the background on the computer system. In the exemplary embodiment depicted in FIG. **1**, the shields **120** are divided into the operating system shields **120**A and the browser shields **120**B. The shields **120** are designed to watch for pestware and for typical pestware activity and includes two types of shields: behavior-monitoring shields and definition-based shields.

As an example, the shields **120** monitor the protected computer **100** for certain types of activities that generally correspond to pestware behavior. Particular examples of some of the types of activities that are monitored include a process spawning another process, an alteration to registry entries, communications with remote sites via the Internet, alterations to a start up folder, injection of a DLL into another process, and a change to the browser's home page and/or bookmarks. Some specific examples of shields are disclosed in the above-identified application entitled System and Method for Locating Malware and Generating Malware Definitions. In the exemplary embodiment, the shields **120** inform the heuristics engine **108** about the activities and the heuristics engine **108** determines whether the research module **108** should be informed and/or whether the activity should be blocked.

As an example, the heuristics module **110** may compare the activities on the computer with weighted factors to make decisions relative to activities at the protected computer. Each of these factors may be, for example, associated with a particular activity and each factor may be weighted by the likelihood that the activity is associated with pestware. If the sum of the weighted factors that match the activity history exceed a threshold, then the activity is identified as pestware activity and the heuristics module **110** prompts the research module **108** to initiate research into the origin of the pestware initiating the activity. It should be recognized that this type of heuristics operation is merely one example, and that the heuristics module **110** may use other techniques to analyze activity on the computer. Additional information related to heuristics-based scanning is found in the above-identified applications entitled System and Method For Heuristic Analysis to Identify Pestware and Client Side Exploit Tracking.

In the exemplary embodiment depicted in FIG. **1**, the time stamp module **112** is configured to send a time stamp to the research module **108** in response to a request from a user and/or in response to the heuristics module **108**. In some embodiments for example, the heuristics module **108** pro-

vides the user with information about suspect activity on the computer **100** so that the user has the option as to whether or not the research module **108** will attempt to identify the source of the activity.

In other embodiments, the heuristics module **110** prompts the time stamp module **112** to initiate the generation of a time stamp, without user intervention, in response to activity that is indicative of pestware activity. In one implementation, for example, the heuristics module **110** prompts the research module **108** to initiate tracing of the pestware activity to an origin of the pestware that is associated with the activity, and the heuristics module **110** also prompts the time stamp module **112** to send a time stamp **112** to the research module **108** so that the research module **108** is provided with a time reference as well as information about the pestware activities.

In yet other embodiments, the timestamp module **112** operates independently of the heuristics module **110**. For example, the timestamp module **112** may prompt the research module **108** to initiate a review of the activity log **121** and/or one or more of the N files in response to a user request.

The activity monitor **116** in several embodiments monitors activities on the computer and stores information about the activities in the activity log **121** so as to assist the research module **108** in identifying activities associated with pestware and the source of the pestware. In some embodiments, for example, the activity log **121** includes a list of processes that are running on the protected computer and the files that are associated with the processes. Although not depicted in FIG. **1**, the shields **120** may utilize the activity monitor **116** to detect pestware-related events and intercept efforts by pestware to spawn new processes or alter aspects of the protected computer **100**.

In some variations, the activity monitor **116** may inject code into existing processes so that when a process attempts to call a function of the computer's **100** operating system (not shown) the injected code can check the process to be started and raise a flag if the existing process is attempting to create a new pestware process or attempting to alter one or more settings (e.g., a registry setting) of the protected computer **100**.

In other embodiments, the activity monitor **116** is realized by a kernel-mode driver that may be loaded during a boot sequence for the protected computer or anytime later. In these embodiments, the activity monitor **116** is configured to log API calls in the activity log **121**. In many variations for example, when a process (e.g., the pestware process) attempts to spawn a another pestware process or alter a registry entry, the API call utilized by the process is intercepted before it is carried out by an operating system of the protected computer. In this way, the attempt may be logged in the activity log **121** and the process may be analyzed to determine whether it is known to be a pestware-related process. Additional details of use of a kernel-level driver in connection with pestware management may be found in the above identified application entitled: System and Method for Kernel-Level Pestware Management.

It should be recognized that the block diagram in FIG. **1** depicts functional capabilities associated with several embodiments of the present invention. One of ordinary skill in the art will recognize that the functions described with reference to FIG. **1** may be realized by various implementations of software in connection with hardware or hardware alone. In these implementations several functions may be consolidated into a single module, for example, and as a consequence, may appear different from the block diagram in FIG. **1** without departing from the scope of the present invention.

Referring next to FIG. **2**, shown are exemplary resources, also referred to herein as historical logs, that are available to the research module **108** in accordance with one embodiment of the present invention. In general, these resources are logs of historical events that occurred on the computer, and each of the logs includes information that may be used to reference other information when, for example, activities, processes and files are traced so as to determine an origin of the pestware or suspected pestware. As shown, a file storage device **218** in this embodiment includes an activity log **220**, which may be generated by an activity monitor (e.g., the activity monitor **116**) and may include information about processes running on the computer **100** and files corresponding to the processes. In variations, the activity log **220** includes a history of API calls and respective times made by processes running on the protected computer, but this is certainly not required.

In addition, the file storage device **218** includes a browser history **240**, browser cache **242**, browser settings **244**, OS settings **246**, an event log **248**, a debugging log **250**, a firewall log **252**, file information **254** and monitoring software logs **256**. One or more of these exemplary files **240-256** may be implemented for one or more of the N files **130** described with reference to FIG. **1**.

The browser history **240** in this embodiment includes a listing of web sites and respective times that the web sites were visited by the user. The browser cache **242** in this embodiment includes files, cookies, and other objects cached in connection with use of a browser of the protected computer (e.g., Internet Explorer or Mozilla (not shown)). As depicted, the browser cache **242** includes a cache index **243** that includes a listing, which associates the content of the browser cache **242** with URLs and time stamps. As discussed further herein, the cache index **243** provides an efficient means for identifying the times objects were retrieved and the URLs that the objects were retrieved from.

The browser settings **244** include information about settings associated with a browser and may include a home page setting and list of user favorites. The browser settings **244** are monitored by the shields **120** for changes. Those settings also contain URLs that may be referenced in time stamped logs, firewall logs, browser histories, etc.

The operating system (OS) settings **246** may include registry entries, a start up folder and other information utilized by an operating system of the protected computer. As discussed further herein, data included in the OS settings **246** may include time stamps, which indicate when changes were made to the settings.

The event log **248** in this embodiment includes a log of events that is maintained by an operating system of the protected computer **100**. For example, the event log **248** may include event information including errors, user log-in history, a listing of processes that have been launched (and the users that launched the processes), path information, information about which process (and which users) accessed a secure area and other information relating to operations of the computer **100**.

Also shown in FIG. **2** is a debugging log **250** that includes application errors, which point to a process and the address where the error occurred. Some techniques employed by pestware forcibly shut down applications, or cause applications to crash when their memory space is injected with pestware code. The infected applications like "explorer.exe" will crash, or some times restart spontaneously. These events/occurrences show up in debugging logs. These are time stamped events, that also reference files on the system.

The firewall log **252** is this embodiment is a collection of information relating to network-related events on the pro-

tected computer. The firewall log **252**, may for example, include time stamps of network activities on the protected computer, which may be utilized by the research module **108** to locate pestware or indicia of pestware.

Also shown in the data storage device **218** is a collection of file information **254**, also known as the file system, which includes a database that the operating system uses to locate and store information about files. For example, the file information **254** may include the date and time a file is created, a date and time the file was last modified, the date and time the files was last accessed and the size of the file.

The monitoring-software logs **256** includes information collected from one or more pieces of monitoring software that are configured to monitor activity on the computer. As an example, the monitoring logs **256** may be generated from a filter driver, a module interfacing with a layer service provider and/or browser helper objects. It should be recognized that the logs **220**, **240**-**256** depicted in FIG. **2** are merely exemplary and these logs are by no means representative all the potential logs that may be accessed to research origins of pestware and/or suspected-pestware.

Referring next to FIG. **3**, shown is a flowchart depicting a method in accordance with one embodiment. Although reference will be made to FIGS. **1** and **2** for exemplary purposes, it should be recognized that the method described with reference to FIG. **3** is certainly not limited to the specific embodiments described with reference to FIGS. **1** and **2**. As shown, in this embodiment a scan of a computer (e.g., the computer **100**) for pestware is initially carried out (Block **302**). In several embodiments for example, the detection engine **102** scans the file storage device **118**, the operating system registry and executable memory of the protected computer for indicia of pestware (e.g., processes or files matching pestware definitions and/or alterations to a registry entry that are consistent with pestware).

As shown in FIG. **3**, if pestware indicia is found, then recorded information (e.g., the activity log **121** and/or the N files **130**) that may include traces of the pestware is accessed (Block **304**), and traversed to search for information leading to the identification of the source of the pestware (Block **306**). In some instances, the origin of the pestware may be identified by simply referencing one piece of data that is stored in connection with the identification of the pestware source. In other instances, however, it may be necessary to access several pieces of referential data, which may be located in one or more of the files **220**-**256** before arriving at the identity of the source of the pestware.

As an example, if a pestware file is found on the file storage device **118**, then the cache index **243** of the browser cache **242** may be searched to identify the name of the file, and if the originating URL is stored in connection with the file, the URL is identified as the source of the file. As another example, if a pestware process is identified, a search of the activity log **220** may lead to the identity of a second process that spawned the pestware process, and additional searching of the activity log **220** using the name of the second process may lead to the identification of a pestware file associated with the second process. In turn, a search of the cache index **243** for the name of the pestware file may lead to the URL from which the pestware file was downloaded.

As shown in FIG. **3**, if the source of the pestware is identified (Block **308**), then the source of the pestware is reported (Block **310**). In some embodiments, it is contemplate that several other computers configured in accordance with the protected computer **100** depicted in FIG. **1** may also report sources of pestware to a centralized location where the URLs may be added to a list of "bad URLs." In addition, the identified URLs may be actively searched to learn more about the pestware generated at the sites, which may help generate definitions for the pestware and may provide information about how pestware infections occur.

Referring next to FIG. **4**, shown is a flowchart depicting a method in accordance with another embodiment of the present invention. Again, reference will be made to FIGS. **1** and **2** for exemplary purposes, but it should be recognized that the method described with reference to FIG. **4** is certainly not limited to the specific embodiments described with reference to FIGS. **1** and **2**. As shown, in this embodiment activity on a computer is monitored for indicia of pestware (Block **402**), and if potential pestware-related activity is detected, recorded information related to the activity is searched to identify one or more suspicious objects (e.g., files and/or processes) that are related to the activity (Block **404**). In some embodiments, the shields **120** (described with reference to FIG. **1**) may monitor the protected computer **100** for activities, and if the activity is identified as potential pestware activity (e.g., by the heuristics module **110**), then the research module **108** searches the activity log **121** and/or one or more of the N files for information relating to the pestware-related activity.

As shown in FIG. **4**, recorded information (e.g., one or more of the N files) is then traversed to trace through information related to the suspicious objects that leads to an origin of the suspicious objects (Block **406**). In one embodiment for example, the suspicious activity leads to a search for suspicious processes and/or files related to the activity (e.g., using the activity log **121**), which then leads to a search of one or more of the N files **130** (e.g., the cache index **243**) for an indication of the source of the suspicious process and/or files.

As depicted in FIG. **4**, if the source of the suspicious object(s) is identified (Block **408**), then the source of the suspicious object(s) is then reported (e.g., to a pestware research entity). In this way, the suspicious objects and the web sites originating the suspicious objects may be further researched to establish the extent to which they may be a threat.

As an example of pestware-related activity that may trigger the search for a source of the activity, if a series of particular API calls is made in a pattern known to be associated with pestware, the process(es) making the calls may be identified using, for example, the activity log **121**. In turn, the activity log **121** may be used to identify the file(s) associated with the process(es), and the cache index **243** may be utilized to search for a source of the file(es). It should be recognized that this is merely one example of the type of activity that may trigger backwards researching of logs on a computer, and that patterns in process creation, downloaded files, changes to an operating system registry and browser settings, for example, may trigger a search of the computer's logs.

Referring next to FIG. **5**, shown is a flowchart depicting yet another method in accordance with another embodiment of the present invention. While referring to FIG. **5**, simultaneous reference will be made to FIGS. **1** and **2** for exemplary purposes, but it should be recognized that the method described with reference to FIG. **5** is certainly not limited to the specific embodiments described with reference to FIGS. **1** and **2**.

As shown, in this embodiment a time of interest is initially established based upon a suspicion that pestware has infected a computer (Block **502**). In some embodiments, for example, a user may establish the time of interest based upon events the user observed (e.g., pop-ups or a system crash). In one embodiment, as discussed with reference to FIG. **1**, in response to a user request, the time stamp module **112** may issue a time stamp and initiate research that is related to activity occurring at or around the time of the time stamp. In

variations, the user is provide with an alert in response to the heuristics module **110** identifying an activity that is suspicious and the user is given an option to initiate research at or around the time of interest.

In other embodiments, the time stamp module **112** automatically generates a time stamp in response to a report of suspicious activity (e.g., from the shields **120** or heuristics module **110**).

As shown in FIG. **5**, once a time of interest is established, suspicious activity is identified on the computer based upon the time of interest (Block **504**). The time of interest may be established, for example, to include a time period before the time stamp is issued so that a search for suspicious activity is limited to a particular time period. In some embodiments, the activity log **120** and/or one or more of the N files **130** are accessed and analyzed to determine whether any activity during the time of interest is suspicious (e.g., the activity indicates in some way that it may be associated with pestware). As an example, if any logged information (e.g., in the activity log **120** and/or one or more of the N files **130**) indicates that during the time of interest that, for example, access to the registry was carried out in connection with a downloaded file or the launch of a process, the activities may identified as being suspect and further research relative to the process and the file may carried out.

Beneficially, many of the logs accessed include time-stamped information, which enables an activity that occurred during the time of interest to be readily identified and analyzed either alone or in connection with other activities occurring during the time of interest. As an example, the activity log **220**, the browser history **240**, browser cache **242**, operating system setting **246**, the event log **248**, the debugging log **250** the firewall log **252**, the file information **254** and the monitoring software logs **256** include time stamped information that provides insight into the activities that occurred on the computer during the time of interest.

As depicted in FIG. **5**, once one or more activities are identified as being suspicious (Block **504**), one or more objects (e.g., processes or files) on the computer are associated with the suspicious activity (Block **506**). For example, the research module **108** may search the activity log **121** and/or one or more of the N files for information that associates the suspicious activity to one or more processes and the processes may be related to one or more files.

As shown in FIG. **5**, recorded information on the computer is then traversed in order to trace to an origin of one or more of the objects (Block **508**). For example, a search of one or more of the N files **130** (e.g., the cache index **243**) may be carried out to identify the source of a suspicious process and/or files. Once the source of the suspicious object(s) is identified (Block **510**), the source is then reported (e.g., to a remote research entity)(Block **512**).

It should be recognized that the methods described with reference to FIGS. **3**, **4** and **5** are merely exemplary and are certainly not the only modes of operation that are contemplated. As an example, the establishment of a time of interest, as discussed with reference to FIG. **5**, may be useful in the method described with reference to FIGS. **3** and **4** for identifying information that leads to the source of the pestware or pestware-related activities. Moreover, it is contemplated that aspects from all three of the methods described with reference to FIGS. **3**, **4** and **5** may be combined.

In conclusion, the present invention provides, among other things, a system and method for identifying a source of pestware or suspected pestware on a computer. Those skilled in the art can readily recognize that numerous variations and substitutions may be made in the invention, its use and its configuration to achieve substantially the same results as achieved by the embodiments described herein. Accordingly, there is no intention to limit the invention to the disclosed exemplary forms. Many variations, modifications and alternative constructions fall within the scope and spirit of the disclosed invention as expressed in the claims.

What is claimed is:

1. A method for identifying an origin of suspected pestware activity on a computer, the method comprising:

monitoring, with a kernel-mode driver, activity on the computer;

generating an activity log on a file storage device of the computer from the kernel-mode driver;

receiving, from a user via an interface of the computer, a time of interest relating to a suspicion of pestware on the computer, wherein the time of interest includes a time interval;

issuing a timestamp after receiving the time of interest;

identifying, based upon the time of interest, indicia of pestware, wherein the identifying is initiated by the issuing the timestamp; and

accessing, using a hardware processor of the computer, at least a portion of a recorded history of externally networked sources that the computer received files from so as to identify, based at least in part upon the identified indicia of pestware, a reference to an identity of an externally networked source that is suspected of originating pestware;

wherein the recorded history of externally networked sources is stored on the file storage device.

2. The method of claim **1**, wherein the identifying includes:

accessing at least one log of historical events on the computer;

locating at least one event in the at least one log that corresponds to the time of interest; and

determining that the at least one event is associated with the indicia of pestware.

3. The method of claim **2**, wherein the determining that the at least one event is associated with the indicia of pestware includes following references in the at least one log to a suspected pestware object.

4. The method of claim **3**, wherein following the references includes following at least one reference in the activity log from the event to a suspected pestware process.

5. The method of claim **2**, wherein the at least one log includes at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

6. The method of claim **1**, wherein the user input is in response to an alert provided to the user regarding a detected suspicious activity on the computer.

7. The method of claim **1**, including:

reporting the identity of the externally networked source to an externally networked pestware research entity so as to enable the pestware research entity to research whether the externally networked source is a source of pestware.

8. The method of claim **1**, wherein the externally networked source is identified by an identifier selected from the group consisting of an I.P. address and a URL.

9. A system for identifying a source of suspected pestware on a computer, the system comprising:

a hardware processor;

a file storage device coupled to the hardware processor;

a timestamp module configured to be executed by the hardware processor, to generate a time of interest including a

time interval in response to detected activity on the computer that is indicative of pestware, to issue a timestamp after generating the time of interest, and to initiate, in response to the issuance of the timestamp, identification of indicia of pestware;

a research portion configured to be executed by the hardware processor and to access at least one log on a file storage device of the computer to relate the time of interest to at least one externally networked source of potential pestware activity on the computer;

a kernel-mode driver configured to monitor activity on the computer and to generate an activity log that is included within the at least one log; and

a reporting portion configured to be executed by the hardware processor and to generate a report that identifies the at least one externally networked source of potential pestware activity.

10. The system of claim **9**, wherein the at least one log includes information that relates the time of interest to an event, and wherein the at least one log includes information that relates the event to a suspected pestware object, and wherein the at least one log includes information that relates the suspected pestware object to the at least one externally networked source of potential pestware activity.

11. The system of claim **10**, including:

a heuristics module configured to be executed by the hardware processor and to evaluate whether the event is a suspected pestware-related event before the research portion accesses information in the at least one log that relates the event to the suspected pestware object.

12. The system of claim **11**, wherein the heuristics module determines that the at least one event is suspected to be pestware-related by analyzing other events in connection with the event, wherein the other events are also related to the time of interest.

13. The system of claim **9**, wherein the at least one log includes at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

14. The system of claim **9**, wherein the timestamp module is further configured to generate the time of interest as a point in time.

15. The system of claim **9**, wherein the at least one externally networked source of potential pestware activity is identified by an identifier selected from the group consisting of an I.P. address and a URL.

16. The system of claim **9**, wherein the at least one log includes at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

* * * * *

Exhibit 10

US008201243B2

(12) **United States Patent**
Boney

(10) **Patent No.: US 8,201,243 B2**
(45) **Date of Patent: Jun. 12, 2012**

(54) **BACKWARDS RESEARCHING ACTIVITY INDICATIVE OF PESTWARE**

(75) Inventor: **Matthew L. Boney**, Longmont, CO (US)

(73) Assignee: **Webroot Inc.**, Broomfield, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1150 days.

(21) Appl. No.: **11/408,146**

(22) Filed: **Apr. 20, 2006**

(65) **Prior Publication Data**

US 2007/0250817 A1 Oct. 25, 2007

(51) **Int. Cl.**
G06F 21/00 (2006.01)

(52) **U.S. Cl.** **726/22**; 713/164; 713/202; 380/279; 717/124; 726/27

(58) **Field of Classification Search** 726/22
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,623,600 A 4/1997 Ji et al.
5,920,696 A 7/1999 Brandt et al.
5,951,698 A 9/1999 Chen et al.
6,069,628 A 5/2000 Farry et al.
6,073,241 A 6/2000 Rosenberg et al.
6,092,194 A 7/2000 Touboul
6,154,844 A 11/2000 Touboul
6,167,520 A 12/2000 Touboul
6,310,630 B1 10/2001 Kulkarni et al.
6,397,264 B1 5/2002 Stasnick et al.
6,405,316 B1 6/2002 Krishnan et al.
6,460,060 B1 10/2002 Maddalozzo, Jr. et al.
6,480,962 B1 11/2002 Touboul
6,535,931 B1 3/2003 Celi, Jr.
6,611,878 B2 8/2003 De Armas et al.
6,633,835 B1 10/2003 Moran et al.
6,667,751 B1 12/2003 Wynn et al.
6,701,441 B1 3/2004 Balasubramaniam et al.
6,772,345 B1 8/2004 Shetty
6,785,732 B1 8/2004 Bates et al.
6,804,780 B1 10/2004 Touboul
6,813,711 B1 11/2004 Dimenstein
6,829,654 B1 12/2004 Jungek

(Continued)

FOREIGN PATENT DOCUMENTS

WO WO 98/45778 10/1998

(Continued)

OTHER PUBLICATIONS

Elements of Computer Security|http://www.google.com/url?sa=t&rct=j&q=&esrc=s&frm=1&source=web&cd=8&ved=0CEwQFjAH&url=http%3A%2F%2Fcms.vle.ase.md%2Fget%2Fpdf%2F7664&ei=_pzyTp3mMeLL0QG7-9zLAg&usg=AFQjCNEmOz7OxykRPyP1_ePiXGgJI1WGWA&sig2=XE_fLI7CTuchuObMdbGYtA|David Salomon|2010.*

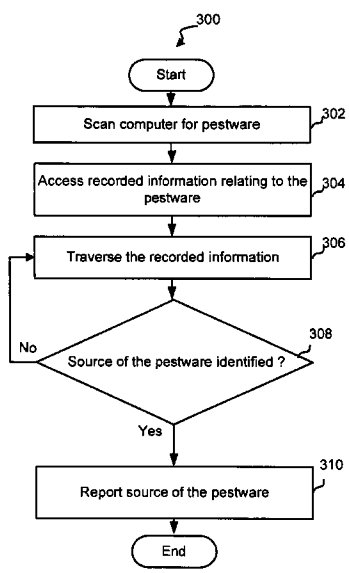
(Continued)

Primary Examiner — Taghi Arani
Assistant Examiner — Mahfuzur Rahman
(74) *Attorney, Agent, or Firm* — Cooley LLP

(57) **ABSTRACT**

A system and method for researching an identity of a source of activity that is indicative of pestware is described. In one embodiment the method comprises monitoring the computer for activity that is indicative of pestware, identifying, based upon the activity, an object residing on the computer that is a suspected pestware object; and accessing at least a portion of a recorded history of sources that the computer received files from so as to identify a reference to an identity of a particular source that the suspected pestware object originated from.

14 Claims, 5 Drawing Sheets



U.S. PATENT DOCUMENTS

6,910,134	B1	6/2005	Maher et al.	
6,965,968	B1	11/2005	Touboul	
7,058,822	B2	6/2006	Edery et al.	
7,058,976	B1 *	6/2006	Dark	726/23
7,107,617	B2	9/2006	Hursey et al.	
7,849,185	B1 *	12/2010	Rockwood	709/224
2002/0078381	A1 *	6/2002	Farley et al.	713/201
2002/0162015	A1	10/2002	Tang	
2002/0162017	A1	10/2002	Sorkin	
2002/0166063	A1	11/2002	Lachman et al.	
2003/0065943	A1 *	4/2003	Geis et al.	713/201
2003/0074581	A1	4/2003	Hursey et al.	
2003/0101381	A1	5/2003	Mateev et al.	
2003/0154399	A1 *	8/2003	Zuk et al.	713/201
2003/0159070	A1 *	8/2003	Mayer et al.	713/201
2003/0217287	A1	11/2003	Kruglenko	
2004/0024864	A1 *	2/2004	Porras et al.	709/224
2004/0030914	A1	2/2004	Kelley et al.	
2004/0034794	A1	2/2004	Mayer et al.	
2004/0064515	A1 *	4/2004	Hockey	709/206
2004/0064736	A1	4/2004	Obrecht et al.	
2004/0080529	A1	4/2004	Wojcik	
2004/0143763	A1	7/2004	Radatti	
2004/0187023	A1	9/2004	Alagna et al.	
2004/0225877	A1 *	11/2004	Huang	713/100
2005/0005160	A1	1/2005	Bates	
2005/0038697	A1	2/2005	Aaron	
2005/0120242	A1 *	6/2005	Mayer et al.	713/201
2005/0138433	A1	6/2005	Linetsky	
2005/0154885	A1	7/2005	Viscomi et al.	
2005/0268112	A1 *	12/2005	Wang et al.	713/188
2006/0085528	A1 *	4/2006	Thomas	709/223
2006/0230290	A1 *	10/2006	Burtscher	713/188
2007/0074289	A1 *	3/2007	Maddaloni	726/23
2007/0094725	A1 *	4/2007	Borders	726/22
2007/0168694	A1 *	7/2007	Maddaloni et al.	714/4
2007/0169198	A1 *	7/2007	Madddaloni et al.	726/24
2007/0220043	A1 *	9/2007	Oliver et al.	707/103 R

FOREIGN PATENT DOCUMENTS

WO	WO 9845778	A2 *	10/1998
WO	PCT/US2006/025378		9/2007
WO	PCT/US2007/067078		11/2007
WO	PCT/US2007/067084		11/2007
WO	PCT/US2006/041798		12/2007

OTHER PUBLICATIONS

PCT/US2007/067076 International Search Report and Written Opinion of the International Searching Authority mailed Nov. 2, 2007.
U.S. Appl. No. 10/956,573, filed Oct. 1, 2004, Steve Thomas.
U.S. Appl. No. 10/956,574, filed Oct. 1, 2004, Steve Thomas.
U.S. Appl. No. 10/956,818, filed Oct. 1, 2004, Matthew Boney.
U.S. Appl. No. 11/257,609, filed Oct. 25, 2005, Michael Burtscher.
U.S. Appl. No. 11/237,291, filed Sep. 28, 2005, Philip Maddaloni.
U.S. Appl. No. 11/408,215, filed Apr. 20, 2006, Matthew Boney.
U.S. Appl. No. 11/408,145, filed Apr. 20, 2006, Matthew Boney.
Codeguru, Three Ways to Inject Your Code Into Another Process, by Robert Kuster, Aug. 4, 2003, 22 pgs.
Codeguru, Managing Low-Level Keyboard Hooks With the Windows API for VB .Net, by Paul Kimmel, Apr. 18, 2004, 10 pgs.
Codeguru, Hooking the Keyboard, by Anoop Thomas, Dec. 13, 2001, 6 pgs.
Illusive Security, Wolves in Sheep's Clothing: malicious DLLs Injected Into trusted Host Applications, Author Unknown, http://home.arcor.de/scheinsicherheit/dll.htm 13 pgs.
DevX.com, Intercepting Systems API Calls, by Seung-Woo Kim, May 13, 2004, 6 pgs.
Microsoft.com, How to Subclass a Window in Windows 95, Article ID 125680, Jul. 11, 2005, 2 pgs.
MSDN, Win32 Hooks by Kyle Marsh, Jul. 29, 1993, 15 pgs.
PCT Search Report, PCT/US05/34874, Jul. 5, 2006, 7 Pages.
Yurcik, William et al., A Planning Framework for Implementing Virtual Private Networks, Jun. 2001, IT Pro, IEEE, pp. 41-44.

* cited by examiner

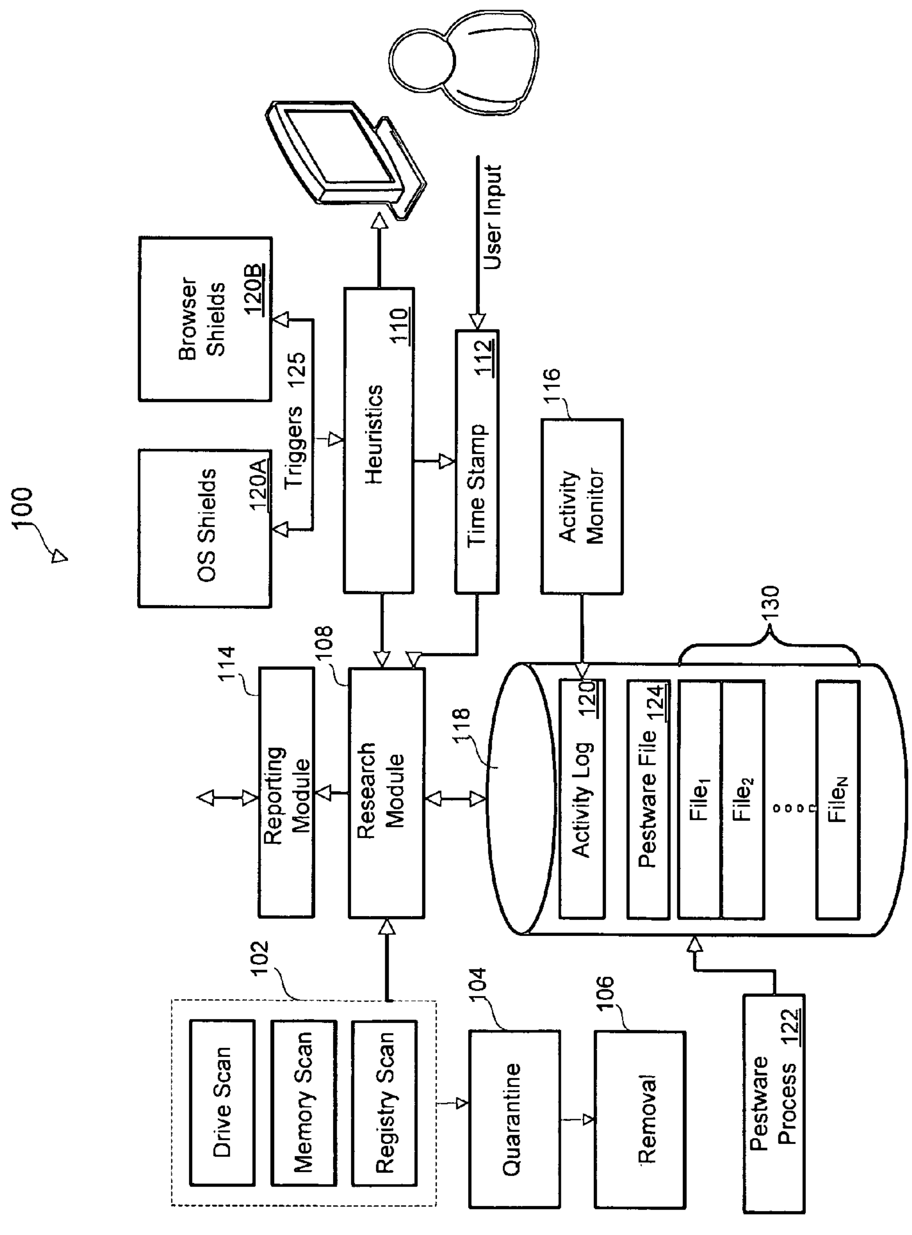

100
OS Shields
120A
Browser Shields
120B
Triggers 125
Heuristics
110
Time Stamp
112
User Input
Activity Monitor
116
114
Reporting Module
108
Research Module
118
130
Activity Log 120
Pestware File 124
File1
File2
FileN
102
Drive Scan
Memory Scan
Registry Scan
104
Quarantine
106
Removal
Pestware Process 122

FIGURE 1

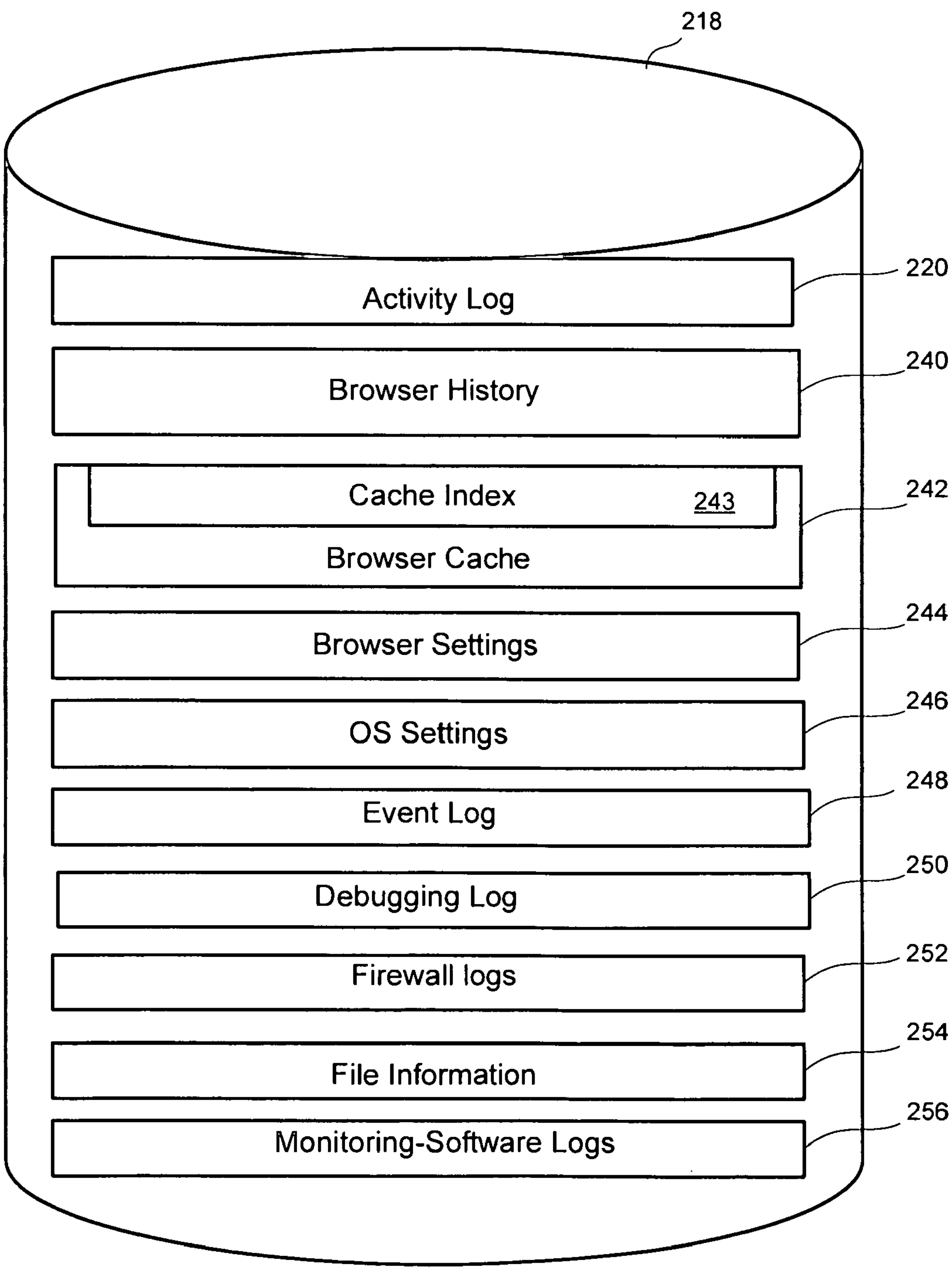

218
Activity Log
220
Browser History
240
Cache Index
243
Browser Cache
242
Browser Settings
244
OS Settings
246
Event Log
248
Debugging Log
250
Firewall logs
252
File Information
254
Monitoring-Software Logs
256

FIGURE 2

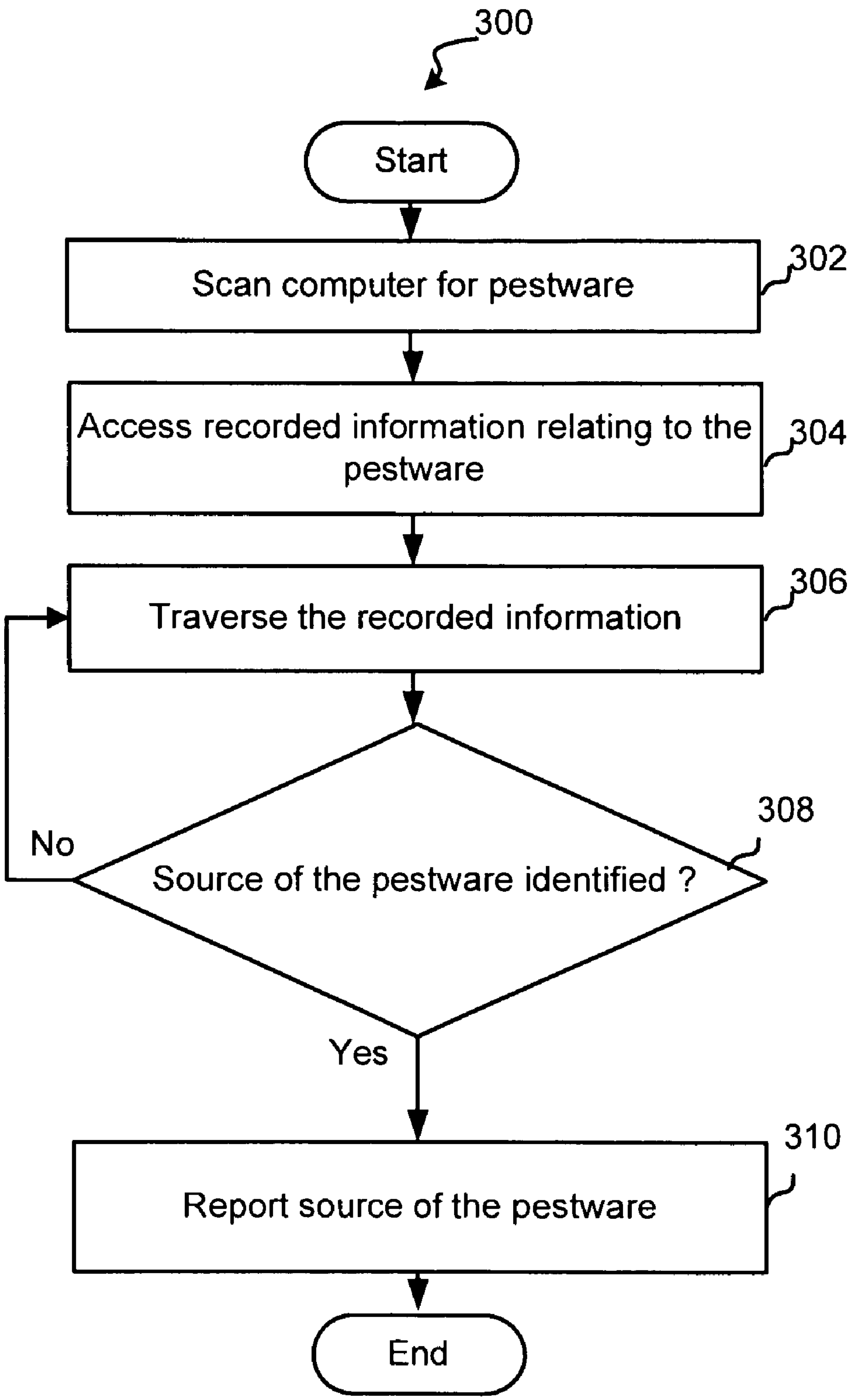

300
Start
Scan computer for pestware
302
Access recorded information relating to the pestware
304
Traverse the recorded information
306
Source of the pestware identified ?
308
No
Yes
Report source of the pestware
310
End

FIGURE 3

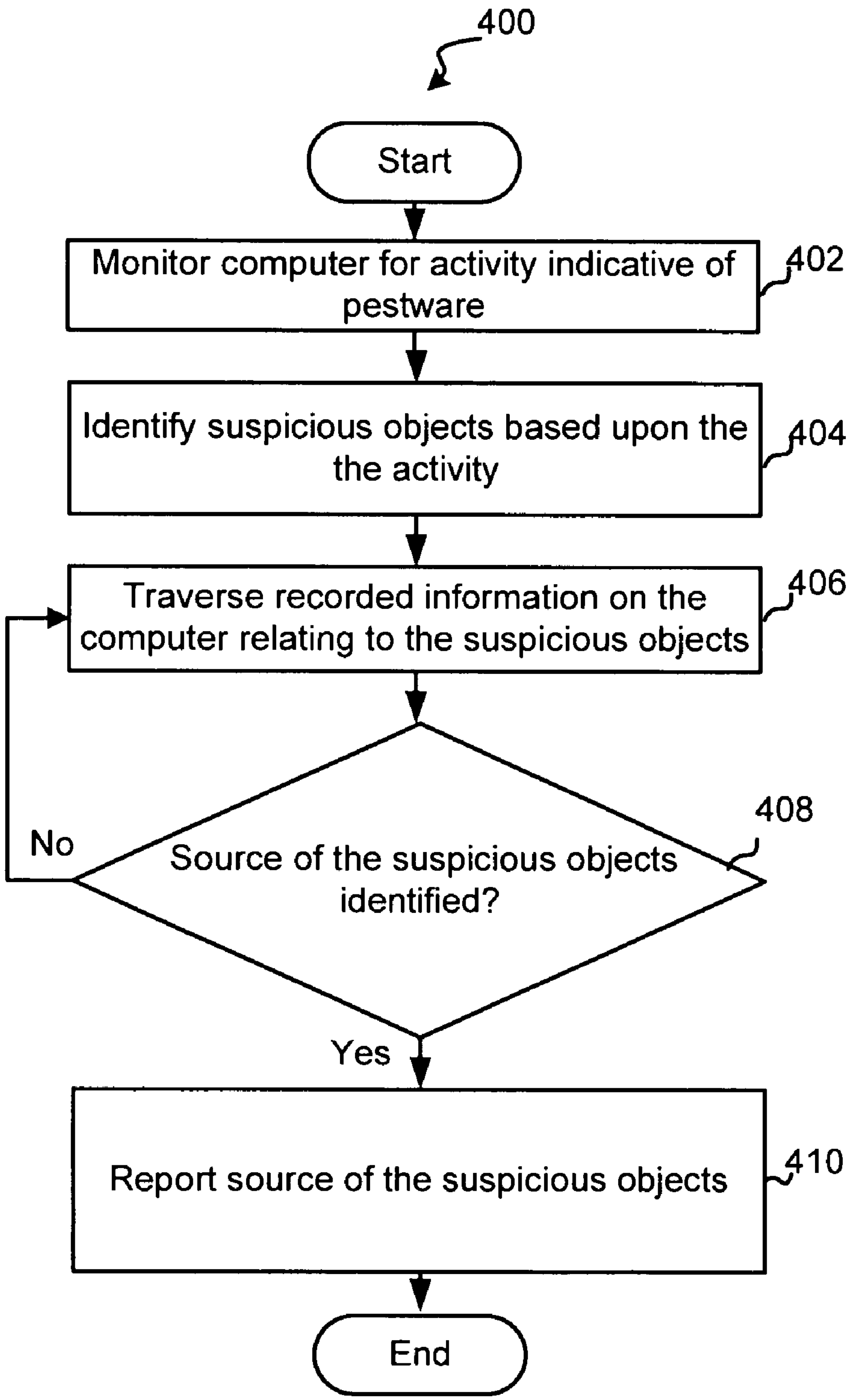

400
Start
Monitor computer for activity indicative of pestware
402
Identify suspicious objects based upon the the activity
404
Traverse recorded information on the computer relating to the suspicious objects
406
No
Source of the suspicious objects identified?
408
Yes
Report source of the suspicious objects
410
End

FIGURE 4

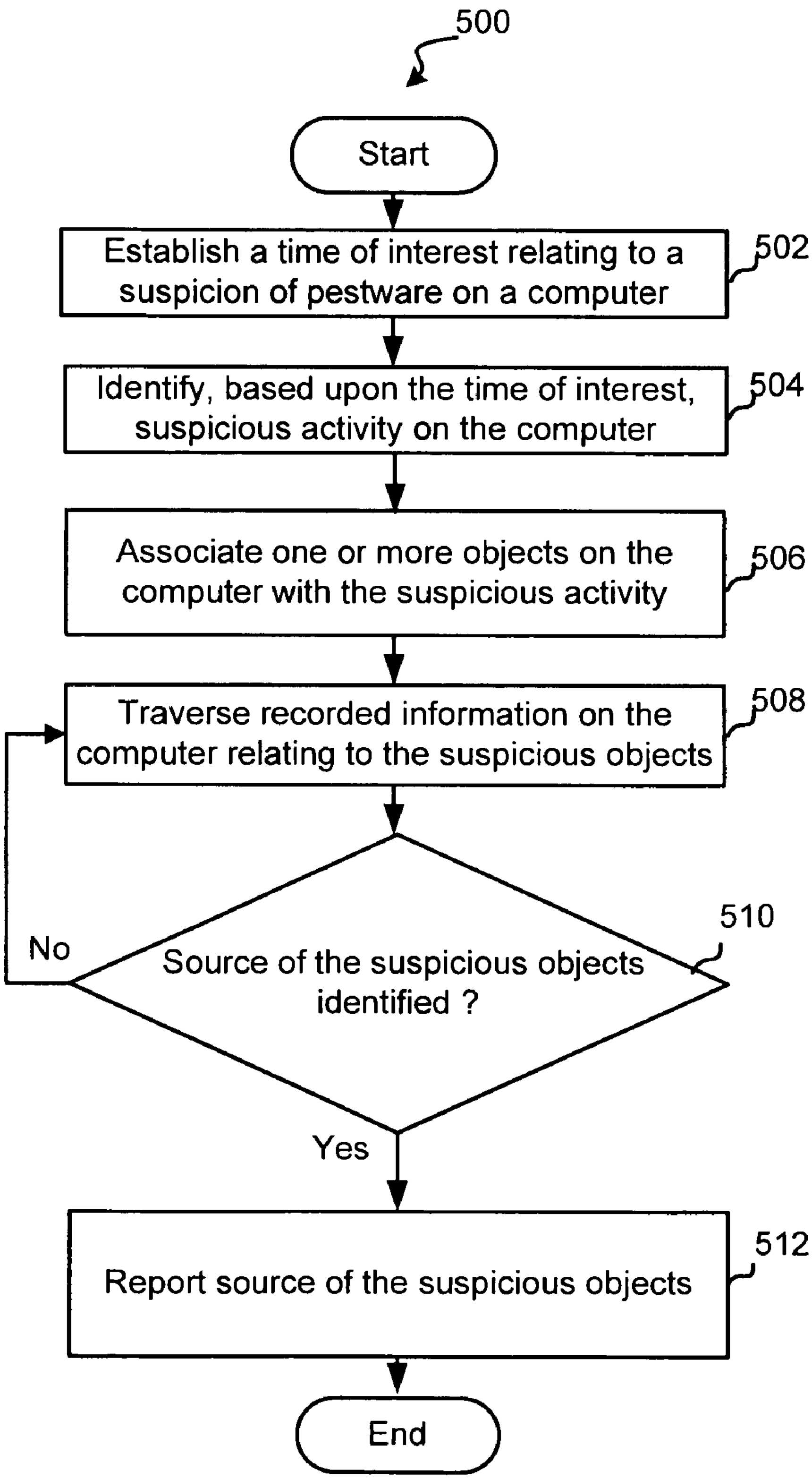

500
Start
Establish a time of interest relating to a suspicion of pestware on a computer
502
Identify, based upon the time of interest, suspicious activity on the computer
504
Associate one or more objects on the computer with the suspicious activity
506
Traverse recorded information on the computer relating to the suspicious objects
508
Source of the suspicious objects identified ?
510
No
Yes
Report source of the suspicious objects
512
End

FIGURE 5

BACKWARDS RESEARCHING ACTIVITY INDICATIVE OF PESTWARE

RELATED APPLICATIONS

The present application is related to commonly owned and assigned application Ser. No. 10/956,573, now U.S. Pat. No. 7,480,683, entitled System and Method For Heuristic Analysis to Identify Pestware; application Ser. No. 10/956,574, now U.S. Pat. No. 7,533,131, entitled System and Method for Pestware Detection and Removal; application Ser. No. 10/956,818, entitled System and Method for Locating Malware and Generating Malware Definitions; application Ser. No. 11/257,609, entitled System and Method for Kernel-Level Pestware Management; application Ser. No. 11/237,291, entitled Client Side Exploit Tracking; application Ser. No. 11/408,215, entitled Backward Researching Existing Pestware, filed herewith; and application Ser. No. 11,408,145, entitled Backward Researching Time Stamped Events to Find an Origin of Pestware, filed herewith, which are incorporated herein by reference.

COPYRIGHT

A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

FIELD OF THE INVENTION

The present invention relates to computer system management. In particular, but not by way of limitation, the present invention relates to systems and methods for detecting, controlling and/or removing pestware.

BACKGROUND OF THE INVENTION

Personal computers and business computers are continually attacked by trojans, spyware, and adware, collectively referred to as "malware," "spyware" or "pestware." These types of programs generally act to gather information about a person or organization—often without the person or organization's knowledge. Some pestware is highly malicious. Other pestware is non-malicious but may cause issues with privacy or system performance. And yet other pestware is actually beneficial or wanted by the user. Unless specified otherwise, "pestware" as used herein refers to any of these programs that collects information about a person or an organization.

Software is presently available to detect and remove pestware. But as it evolves, the software to detect and remove it must also evolve. Accordingly, current techniques and software for removing pestware are not always satisfactory and will most certainly not be satisfactory in the future. Additionally, because some pestware is actually valuable to a user, pestware-detection software should, in some cases, be able to handle differences between wanted and unwanted pestware.

Current pestware removal software uses definitions of known pestware to search for and remove files on a protected system. These definitions are often slow and cumbersome to create. Additionally, it is often difficult to initially locate the pestware in order to create the definitions. Accordingly, a system and method are needed to address the shortfalls of present technology and to provide other new and innovative features.

SUMMARY OF THE INVENTION

Exemplary embodiments of the present invention that are shown in the drawings are summarized below. These and other embodiments are more fully described in the Detailed Description section. It is to be understood, however, that there is no intention to limit the invention to the forms described in this Summary of the Invention or in the Detailed Description. One skilled in the art can recognize that there are numerous modifications, equivalents and alternative constructions that fall within the spirit and scope of the invention as expressed in the claims.

The present invention can provide a system and method for researching an identity of a source of activity that is indicative of pestware. In one embodiment the method comprises monitoring the computer for activity that is indicative of pestware, identifying, based upon the activity, an object residing on the computer that is a suspected pestware object and accessing at least a portion of a recorded history of sources that the computer received files from so as to identify a reference to an identity of a particular source that the suspected pestware object originated from.

As previously stated, the above-described embodiments and implementations are for illustration purposes only. Numerous other embodiments, implementations, and details of the invention are easily recognized by those of skill in the art from the following descriptions and claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Various objects and advantages and a more complete understanding of the present invention are apparent and more readily appreciated by reference to the following Detailed Description and to the appended claims when taken in conjunction with the accompanying Drawings wherein:

FIG. **1** illustrates a block diagram of one implementation of the present invention;

FIG. **2** is a block diagram depicting an embodiment of the file storage device of FIG. **1**;

FIG. **3** is a flowchart depicting a method according to an exemplary embodiment;

FIG. **4** is a flowchart depicting another method according to another embodiment;

FIG. **5** is a flowchart depicting yet another method according to yet another embodiment.

DETAILED DESCRIPTION

Referring now to the drawings, where like or similar elements are designated with identical reference numerals throughout the several views, and referring in particular to FIG. **1**, it illustrates a block diagram of one implementation of the present invention. Shown is a protected computer **100** that includes a detection module **102**, quarantine module **104**, removal module **106**, and shields **120**. In addition, a research module **108** is shown coupled to a heuristics module **110**, a time stamp module **112** and a reporting module **114**.

Each of these modules can be implemented in software or hardware. And if implemented in software, the modules can be implemented in a single software package or in multiple software packages. In addition, one of ordinary skill in the art will recognize that the software can be designed to operate on any type of computer system including WINDOWS and

Linux-based systems. Additionally, the software can be configured to operate on personal computers and/or servers. For convenience, embodiments of the present invention are generally described herein with relation to WINDOWS-based systems. Those of skill in the art can easily adapt these implementations for other types of operating systems or computer systems.

Also shown is a file storage device **118** that is coupled to the research module **108** and an activity monitor **116**. In this embodiment the file storage device includes an activity log **120**, a pestware file **124** and a collection of N files **130**. The file storage device **118** is described herein in several implementations as hard disk drive for convenience, but this is certainly not required, and one of ordinary skill in the art will recognize that other storage media may be utilized without departing from the scope of the present invention. In addition, one of ordinary skill in the art will recognize that the storage device **118**, which is depicted for convenience as a single storage device, may be realized by multiple (e.g., distributed) storage devices.

In the exemplary embodiment, the pestware file **124**, corresponds to (e.g., includes data relating to) a pestware process **122** operating in memory. The pestware process **122** is exemplary of pestware processes that are configured to make one or more unauthorized alterations to the computer **100**. For example, the pestware process **122** may make changes to either or both of the browser settings and/or operating system (OS) settings without approval and/or the knowledge of the user.

In accordance with several embodiments, the research module **108** is configured to receive an indication that either known pestware is residing on the computer or activities indicative of pestware have occurred or are occurring on the protected computer. In response, the research module **108** is configured to research the activity log **120** and/or the N files **130**, which include information (e.g., historical logs) relating to events on the computer **100**, to identify a source of the pestware **122**, **124** or pestware-related activities, which may be reported by the reporting module **114** to a centralized data store for subsequent pestware management purposes.

For example, the identities (e.g., I.P. address, URL, email client or program name) of sources (e.g., web sites, email or program) of the pestware may be collected in a centralized data store (not shown) and then subsequently reported to other users. In addition, the sources of the pestware or suspected pestware may be visited to further research how the pestware is being distributed from the URLs and/or to generate new definitions for pestware discovered at these sources.

As described further with reference to FIG. **2**, the N files **130** include one or more files with information that assist the research module **108** in tracing information in the N files **130** to an identity (e.g., URL) of the source of pestware **122**, **124** on the computer **100**. One or more of the N files **130** may be associated with an operating system of the protected computer and/or one or more applications of the protected computer, and may include information such as process IDs, registry entries, file names, cookies and URLs among other information that is used to trace from an identified pestware file, pestware process and/or pestware activity to an originating source (e.g., URL or IP address) of the infection. In one embodiment, one or more of the N files **130** is generated from an application that generates a log of data after examining the computer. An example of such an application is an application distributed under the name HijackThis.

In the exemplary embodiment depicted in FIG. **1**, the research module **108** is configured to receive indications that pestware may be present on the computer from each of the detection engine **102**, the heuristics engine **110** and the time stamp module **112**, but this is certainly not required. In other embodiments, for example, the research module **108** may be configured to communicate only with the detection engine **102** or only with the heuristics engine. In yet other embodiments, the research module **108** may be configured to receive only a time stamp from the time stamp module **112**.

According to several embodiments, pestware-detection functions operating on the protected computer **100** are represented by the detection engine **102**, the quarantine engine **104**, the removal engine **106**, the shields **120** and the heuristic engine **110**. The basic functions of the detection engine **102** is to compare files, processes and registry entries on the protected computer against known pestware definitions and characteristics. When a match is found, in addition to quarantining and removing a pestware object, the detection engine **102** informs the research module **108** of the pestware. Details associated with several embodiments of sweep, quarantine, and removal engines are found in the above-identified application entitled System and Method for Pestware Detection and Removal.

Pestware and pestware activity can also be identified by the shields **120**, which generally run in the background on the computer system. In the exemplary embodiment depicted in FIG. **1**, the shields **120** are divided into the operating system shields **120**A and the browser shields **120**B. The shields **120** are designed to watch for pestware and for typical pestware activity and includes two types of shields: behavior-monitoring shields and definition-based shields.

As an example, the shields **120** monitor the protected computer **100** for certain types of activities that generally correspond to pestware behavior. Particular examples of some of the types of activities that are monitored include a process spawning another process, an alteration to registry entries, communications with remote sites via the Internet, alterations to a start up folder, injection of a DLL into another process, and a change to the browser's home page and/or bookmarks. Some specific examples of shields are disclosed in the above-identified application entitled System and Method for Locating Malware and Generating Malware Definitions. In the exemplary embodiment, the shields **120** inform the heuristics engine **108** about the activities and the heuristics engine **108** determines whether the research module **108** should be informed and/or whether the activity should be blocked.

As an example, the heuristics module **110** may compare the activities on the computer with weighted factors to make decisions relative to activities at the protected computer. Each of these factors may be, for example, associated with a particular activity and each factor may be weighted by the likelihood that the activity is associated with pestware. If the sum of the weighted factors that match the activity history exceed a threshold, then the activity is identified as pestware activity and the heuristics module **110** prompts the research module **108** to initiate research into the origin of the pestware initiating the activity. It should be recognized that this type of heuristics operation is merely one example, and that the heuristics module **110** may use other techniques to analyze activity on the computer. Additional information related to heuristics-based scanning is found in the above-identified applications entitled System and Method For Heuristic Analysis to Identify Pestware and Client Side Exploit Tracking.

In the exemplary embodiment depicted in FIG. **1**, the time stamp module **112** is configured to send a time stamp to the research module **108** in response to a request from a user and/or in response to the heuristics module **108**. In some embodiments for example, the heuristics module **108** pro-

vides the user with information about suspect activity on the computer **100** so that the user has the option as to whether or not the research module **108** will attempt to identify the source of the activity.

In other embodiments, the heuristics module **110** prompts the time stamp module **112** to initiate the generation of a time stamp, without user intervention, in response to activity that is indicative of pestware activity. In one implementation, for example, the heuristics module **110** prompts the research module **108** to initiate tracing of the pestware activity to an origin of the pestware that is associated with the activity, and the heuristics module **110** also prompts the time stamp module **112** to send a time stamp **112** to the research module **108** so that the research module **108** is provided with a time reference as well as information about the pestware activities.

In yet other embodiments, the timestamp module **112** operates independently of the heuristics module **110**. For example, the timestamp module **112** may prompt the research module **108** to initiate a review of the activity log **120** and/or one or more of the N files in response to a user request.

The activity monitor **116** in several embodiments monitors activities on the computer and stores information about the activities in the activity log **120** so as to assist the research module **108** in identifying activities associated with pestware and the source of the pestware. In some embodiments, for example, the activity log **120** includes a list of processes that are running on the protected computer and the files that are associated with the processes. Although not depicted in FIG. **1**, the shields **120** may utilize the activity monitor **116** to detect pestware-related events and intercept efforts by pestware to spawn new processes or alter aspects of the protected computer **100**.

In some variations, the activity monitor **116** may inject code into existing processes so that when a process attempts to call a function of the computer's **100** operating system (not shown) the injected code can check the process to be started and raise a flag if the existing process is attempting to create a new pestware process or attempting to alter one or more settings (e.g., a registry setting) of the protected computer **100**.

In other embodiments, the activity monitor **116** is realized by a kernel-mode driver that may be loaded during a boot sequence for the protected computer or anytime later. In these embodiments, the activity monitor **116** is configured to log API calls in the activity log. In many variations for example, when a process (e.g., the pestware process) attempts to spawn a another pestware process or alter a registry entry, the API call utilized by the process is intercepted before it is carried out by an operating system of the protected computer. In this way, the attempt may be logged in the activity log **120** and the process may be analyzed to determine whether it is known to be a pestware-related process. Additional details of use of a kernel-level driver in connection with pestware management may be found in the above identified application entitled: System and Method for Kernel-Level Pestware Management.

It should be recognized that the block diagram in FIG. **1** depicts functional capabilities associated with several embodiments of the present invention. One of ordinary skill in the art will recognize that the functions described with reference to FIG. **1** may be realized by various implementations of software in connection with hardware or hardware alone. In these implementations several functions may be consolidated into a single module, for example, and as a consequence, may appear different from the block diagram in FIG. **1** without departing from the scope of the present invention.

Referring next to FIG. **2**, shown are exemplary resources, also referred to herein as historical logs, that are available to the research module **108** in accordance with one embodiment of the present invention. In general, these resources are logs of historical events that occurred on the computer, and each of the logs includes information that may be used to reference other information when, for example, activities, processes and files are traced so as to determine an origin of the pestware or suspected pestware. As shown, a file storage device **218** in this embodiment includes an activity log **220**, which may be generated by an activity monitor (e.g., the activity monitor **116**) and may include information about processes running on the computer **100** and files corresponding to the processes. In variations, the activity log **220** includes a history of API calls and respective times made by processes running on the protected computer, but this is certainly not required.

In addition, the file storage device **218** includes a browser history **240**, browser cache **242**, browser settings **244**, OS settings **246**, an event log **248**, a debugging log **250**, a firewall log **252**, file information **254** and monitoring software logs **256**. One or more of these exemplary files **240-256** may be implemented for one or more of the N files **130** described with reference to FIG. **1**.

The browser history **240** in this embodiment includes a listing of web sites and respective times that the web sites were visited by the user. The browser cache **242** in this embodiment includes files, cookies, and other objects cached in connection with use of a browser of the protected computer (e.g., Internet Explorer or Mozilla (not shown)). As depicted, the browser cache **242** includes a cache index **243** that includes a listing, which associates the content of the browser cache **242** with URLs and time stamps. As discussed further herein, the cache index **243** provides an efficient means for identifying the times objects were retrieved and the URLs that the objects were retrieved from.

The browser settings **244** include information about settings associated with a browser and may include a home page setting and list of user favorites. The browser settings **244** are monitored by the shields **120** for changes. Those settings also contain URLs that may be referenced in time stamped logs, firewall logs, browser histories, etc.

The operating system (OS) settings **246** may include registry entries, a start up folder and other information utilized by an operating system of the protected computer. As discussed further herein, data included in the OS settings **246** may include time stamps, which indicate when changes were made to the settings.

The event log **248** in this embodiment includes a log of events that is maintained by an operating system of the protected computer **100**. For example, the event log **248** may include event information including errors, user log-in history, a listing of processes that have been launched (and the users that launched the processes), path information, information about which process (and which users) accessed a secure area and other information relating to operations of the computer **100**.

Also shown in FIG. **2** is a debugging log **250** that includes application errors, which point to a process and the address where the error occurred. Some techniques employed by pestware forcibly shut down applications, or cause applications to crash when their memory space is injected with pestware code. The infected applications like "explorer.exe" will crash, or some times restart spontaneously. These events/occurrences show up in debugging logs. These are time stamped events, that also reference files on the system.

The firewall log **252** is this embodiment is a collection of information relating to network-related events on the pro-

tected computer. The firewall log **252**, may for example, include time stamps of network activities on the protected computer, which may be utilized by the research module **108** to locate pestware or indicia of pestware.

Also shown in the data storage device **218** is a collection of file information **254**, also known as the file system, which includes a database that the operating system uses to locate and store information about files. For example, the file information **254** may include the date and time a file is created, a date and time the file was last modified, the date and time the files was last accessed and the size of the file.

The monitoring-software logs **256** includes information collected from one or more pieces of monitoring software that are configured to monitor activity on the computer. As an example, the monitoring logs **256** may be generated from a filter driver, a module interfacing with a layer service provider and/or browser helper objects. It should be recognized that the logs **220**, **240**-**256** depicted in FIG. **2** are merely exemplary and these logs are by no means representative all the potential logs that may be accessed to research origins of pestware and/or suspected-pestware.

Referring next to FIG. **3**, shown is a flowchart depicting a method in accordance with one embodiment. Although reference will be made to FIGS. **1** and **2** for exemplary purposes, it should be recognized that the method described with reference to FIG. **3** is certainly not limited to the specific embodiments described with reference to FIGS. **1** and **2**. As shown, in this embodiment a scan of a computer (e.g., the computer **100**) for pestware is initially carried out (Block **302**). In several embodiments for example, the detection engine **102** scans the file storage device **118**, the operating system registry and executable memory of the protected computer for indicia of pestware (e.g., processes or files matching pestware definitions and/or alterations to a registry entry that are consistent with pestware).

As shown in FIG. **3**, if pestware indicia is found, then recorded information (e.g., the activity log **120** and/or the N files **130**) that may include traces of the pestware is accessed (Block **304**), and traversed to search for information leading to the identification of the source of the pestware (Block **306**). In some instances, the origin of the pestware may be identified by simply referencing one piece of data that is stored in connection with the identification of the pestware source. In other instances, however, it may be necessary to access several pieces of referential data, which may be located in one or more of the files **220**-**256** before arriving at the identity of the source of the pestware.

As an example, if a pestware file is found on the file storage device **118**, then the cache index **243** of the browser cache **242** may be searched to identify the name of the file, and if the originating URL is stored in connection with the file, the URL is identified as the source of the file. As another example, if a pestware process is identified, a search of the activity log **220** may lead to the identity of a second process that spawned the pestware process, and additional searching of the activity log **220** using the name of the second process may lead to the identification of a pestware file associated with the second process. In turn, a search of the cache index **243** for the name of the pestware file may lead to the URL from which the pestware file was downloaded.

As shown in FIG. **3**, if the source of the pestware is identified (Block **308**), then the source of the pestware is reported (Block **310**). In some embodiments, it is contemplate that several other computers configured in accordance with the protected computer **100** depicted in FIG. **1** may also report sources of pestware to a centralized location where the URLs may be added to a list of "bad URLs." In addition, the identified URLs may be actively searched to learn more about the pestware generated at the sites, which may help generate definitions for the pestware and may provide information about how pestware infections occur.

Referring next to FIG. **4**, shown is a flowchart depicting a method in accordance with another embodiment of the present invention. Again, reference will be made to FIGS. **1** and **2** for exemplary purposes, but it should be recognized that the method described with reference to FIG. **4** is certainly not limited to the specific embodiments described with reference to FIGS. **1** and **2**. As shown, in this embodiment activity on a computer is monitored for indicia of pestware (Block **402**), and if potential pestware-related activity is detected, recorded information related to the activity is searched to identify one or more suspicious objects (e.g., files and/or processes) that are related to the activity (Block **404**). In some embodiments, the shields **120** (described with reference to FIG. **1**) may monitor the protected computer **100** for activities, and if the activity is identified as potential pestware activity (e.g., by the heuristics module **110**), then the research module **108** searches the activity log **120** and/or one or more of the N files for information relating to the pestware-related activity.

As shown in FIG. **4**, recorded information (e.g., one or more of the N files) is then traversed to trace through information related to the suspicious objects that leads to an origin of the suspicious objects (Block **406**). In one embodiment for example, the suspicious activity leads to a search for suspicious processes and/or files related to the activity (e.g., using the activity log **120**), which then leads to a search of one or more of the N files **130** (e.g., the cache index **243**) for an indication of the source of the suspicious process and/or files.

As depicted in FIG. **4**, if the source of the suspicious object(s) is identified (Block **408**), then the source of the suspicious object(s) is then reported (e.g., to a pestware research entity). In this way, the suspicious objects and the web sites originating the suspicious objects may be further researched to establish the extent to which they may be a threat.

As an example of pestware-related activity that may trigger the search for a source of the activity, if a series of particular API calls is made in a pattern known to be associated with pestware, the process(es) making the calls may be identified using, for example, the activity log **120**. In turn, the activity log **120** may be used to identify the file(s) associated with the process(es), and the cache index **243** may be utilized to search for a source of the file(es). It should be recognized that this is merely one example of the type of activity that may trigger backwards researching of logs on a computer, and that patterns in process creation, downloaded files, changes to an operating system registry and browser settings, for example, may trigger a search of the computer's logs.

Referring next to FIG. **5**, shown is a flowchart depicting yet another method in accordance with another embodiment of the present invention. While referring to FIG. **5**, simultaneous reference will be made to FIGS. **1** and **2** for exemplary purposes, but it should be recognized that the method described with reference to FIG. **5** is certainly not limited to the specific embodiments described with reference to FIGS. **1** and **2**.

As shown, in this embodiment a time of interest is initially established based upon a suspicion that pestware has infected a computer (Block **502**). In some embodiments, for example, a user may establish the time of interest based upon events the user observed (e.g., pop-ups or a system crash). In one embodiment, as discussed with reference to FIG. **1**, in response to a user request, the time stamp module **112** may issue a time stamp and initiate research that is related to activity occurring at or around the time of the time stamp. In

variations, the user is provide with an alert in response to the heuristics module **110** identifying an activity that is suspicious and the user is given an option to initiate research at or around the time of interest.

In other embodiments, the time stamp module **112** automatically generates a time stamp in response to a report of suspicious activity (e.g., from the shields **120** or heuristics module **110**).

As shown in FIG. **5**, once a time of interest is established, suspicious activity is identified on the computer based upon the time of interest (Block **504**). The time of interest may be established, for example, to include a time period before the time stamp is issued so that a search for suspicious activity is limited to a particular time period. In some embodiments, the activity log **130** and/or one or more of the N files **130** are accessed and analyzed to determine whether any activity during the time of interest is suspicious (e.g., the activity indicates in some way that it may be associated with pestware). As an example, if any logged information (e.g., in the activity log **130** and/or one or more of the N files **130**) indicates that during the time of interest that, for example, access to the registry was carried out in connection with a downloaded file or the launch of a process, the activities may identified as being suspect and further research relative to the process and the file may carried out.

Beneficially, many of the logs accessed include time-stamped information, which enables an activity that occurred during the time of interest to be readily identified and analyzed either alone or in connection with other activities occurring during the time of interest. As an example, the activity log **220**, the browser history **240**, browser cache **242**, operating system setting **246**, the event log **248**, the debugging log **250** the firewall log **252**, the file information **254** and the monitoring software logs **256** include time stamped information that provides insight into the activities that occurred on the computer during the time of interest.

As depicted in FIG. **5**, once one or more activities are identified as being suspicious (Block **504**), one or more objects (e.g., processes or files) on the computer are associated with the suspicious activity (Block **506**). For example, the research module **108** may search the activity log **120** and/or one or more of the N files for information that associates the suspicious activity to one or more processes and the processes may be related to one or more files.

As shown in FIG. **5**, recorded information on the computer is then traversed in order to trace to an origin of one or more of the objects (Block **508**). For example, a search of one or more of the N files **130** (e.g., the cache index **243**) may be carried out to identify the source of a suspicious process and/or files. Once the source of the suspicious object(s) is identified (Block **510**), the source is then reported (e.g., to a remote research entity)(Block **512**).

It should be recognized that the methods described with reference to FIGS. **3**, **4** and **5** are merely exemplary and are certainly not the only modes of operation that are contemplated. As an example, the establishment of a time of interest, as discussed with reference to FIG. **5**, may be useful in the method described with reference to FIGS. **3** and **4** for identifying information that leads to the source of the pestware or pestware-related activities. Moreover, it is contemplated that aspects from all three of the methods described with reference to FIGS. **3**, **4** and **5** may be combined.

In conclusion, the present invention provides, among other things, a system and method for identifying a source of pestware or suspected pestware on a computer. Those skilled in the art can readily recognize that numerous variations and substitutions may be made in the invention, its use and its configuration to achieve substantially the same results as achieved by the embodiments described herein. Accordingly, there is no intention to limit the invention to the disclosed exemplary forms. Many variations, modifications and alternative constructions fall within the scope and spirit of the disclosed invention as expressed in the claims.

What is claimed is:

1. A method for identifying an origin of activity on a computer that is indicative of pestware comprising:

monitoring, using a kernel-mode driver, the computer for activity that is indicative of pestware, wherein the monitoring includes monitoring API calls and storing a history of at least a portion of the API calls in an activity log;

analyzing, heuristically, computer activity to determine whether one or more weighted factors associated with an activity exceeds a threshold so as to arrive at a determination that the activity is indicative of pestware;

identifying, based upon the activity, an object residing on the computer that is a suspected pestware object;

accessing, in response to the identifying an object, at least a portion of a recorded history of externally networked sources that the computer received files from so as to identify a reference to an identity of a particular externally networked source that the suspected pestware object originated from; and

reporting the identity of the particular externally networked source to an externally networked pestware research entity so as to enable the externally networked pestware research entity to research whether the particular externally networked source is a source of pestware.

2. The method of claim **1**, wherein the particular externally networked source is identified by an identifier selected from the group consisting of an I.P. address and a URL.

3. The method of claim **2**, wherein the identifying includes accessing an activity log that includes information that relates the activity to the suspected pestware object.

4. The method of claim **1**, wherein the recorded history resides in a browser cache, which includes information about files downloaded to the computer and a source of each of the files.

5. The method of claim **1**, wherein the recorded history resides in at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

6. A system for identifying a source of activity on a computer that is indicative of pestware including:

an activity monitor configured to monitor API calls and to store a history of at least a portion of the API calls in an activity log, wherein the activity monitor includes a kernel-mode driver adapted to intercept the API calls;

a heuristics module configured to identify an activity on the computer that is indicative of pestware residing on the computer and to analyze the activity to determine whether one or more weighted factors associated with the activity exceeds a threshold; and

a research portion configured to access, in response to a prompt from the heuristics module, a first set of recorded information on the computer that relates the activity to at least one file residing on the computer, and wherein the research portion is configured to access a second set of recorded information on the computer that relates the at least one file to an externally networked source from which the file was received; and

a reporting portion configured to generate a report that identifies the externally networked source of the file and

to report an identity of the externally networked source to an externally networked pestware research entity so as to enable the externally networked pestware research entity to research whether the externally networked source is a source of pestware.

7. The system of claim **6**, wherein activity monitor is configured to store an identity of each process that made each of the API calls in the stored history, and wherein the activity monitor is configured to store an identity of each file that corresponds to each the processes that made the API calls in the stored history so as to create a relation between each API call and at least one file.

8. The system of claim **6**, wherein the heuristics module is configured to receive information about the API calls from the activity monitor and determine whether the API calls are indicative of pestware.

9. The system of claim **6**, wherein the first set of recorded information and the second set of recorded information reside in the same log file on a file storage device of the computer.

10. The system of claim **6**, wherein the externally networked source is identified by an identifier selected from the group consisting of an I.P. address and a URL.

11. The system of claim **6** wherein the recorded history resides in at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

12. A non-transitory computer-readable medium including processor-executable instructions for identifying an origin of activity on a computer that is indicative of pestware, the instructions including instructions for:

monitoring, with a kernel-mode driver, the computer for activity that is indicative of pestware, wherein the instructions for monitoring include instructions for monitoring API calls and storing a history of at least a portion of the API calls in an activity log;

analyzing, heuristically, computer activity to determine whether one or more weighted factors associated with an activity exceeds a threshold so as to arrive at a determination that the activity is indicative of pestware;

identifying, based upon the activity, an object residing on the computer that is a suspected pestware object;

accessing, in response to the identifying an object, at least a portion of a recorded history of externally networked sources that the computer received files from so as to identify a reference to an identity of a particular externally networked source that the suspected pestware object originated from; and

reporting the identity of the particular externally networked source to an externally networked pestware research entity so as to enable the externally networked pestware research entity to research whether the particular externally networked source is a source of pestware.

13. The non-transitory computer-readable medium of claim **12**, wherein the particular externally networked source is identified by an identifier selected from the group consisting of an I.P. address and a URL.

14. The non-transitory computer-readable medium of claim **12**, wherein the recorded history resides in at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

* * * * *

Exhibit 11

US008856505B2

(12) United States Patent
Schneider

(10) **Patent No.: US 8,856,505 B2**
(45) **Date of Patent: *Oct. 7, 2014**

(54) **MALWARE MANAGEMENT THROUGH KERNEL DETECTION DURING A BOOT SEQUENCE**

(75) Inventor: **Jerome L. Schneider**, Boulder, CO (US)

(73) Assignee: **Webroot Inc.**, Broomfield, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 24 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **13/460,655**

(22) Filed: **Apr. 30, 2012**

(65) **Prior Publication Data**

US 2012/0216027 A1 Aug. 23, 2012

Related U.S. Application Data

(63) Continuation of application No. 11/462,827, filed on Aug. 7, 2006, now Pat. No. 8,190,868.

(51) **Int. Cl.**
G06F 21/00 (2013.01)
G06F 21/56 (2013.01)
G06F 21/55 (2013.01)

(52) **U.S. Cl.**
CPC ***G06F 21/566*** (2013.01); ***G06F 21/56*** (2013.01); ***G06F 21/554*** (2013.01)
USPC **713/2**; 713/193

(58) **Field of Classification Search**
CPC G06F 21/56; G06F 21/566; G06F 21/554
USPC 713/2, 193
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2003/0135791 A1* 7/2003 Natvig 714/38
2005/0114687 A1* 5/2005 Zimmer et al. 713/193
2005/0125687 A1* 6/2005 Townsend et al. 713/200
2005/0216759 A1* 9/2005 Rothman et al. 713/200

FOREIGN PATENT DOCUMENTS

GB WO2006077443 * 7/2006

OTHER PUBLICATIONS

Wen, "Implicit Detection od Hidden Processes with a Local-Booted Virtual Machinr", 2008, IEEE, p. 150-155.*
Wen, "Implicit Detection of Hidden Processes with a Local-Booted Virtual Machine", 2008, IEEE, p. 150-155 Bai, "Approach for Malware Identification using Dynamic Behavior and Outcome Triggering", 2014, IEEE, vol. 8, p. 140-151.*

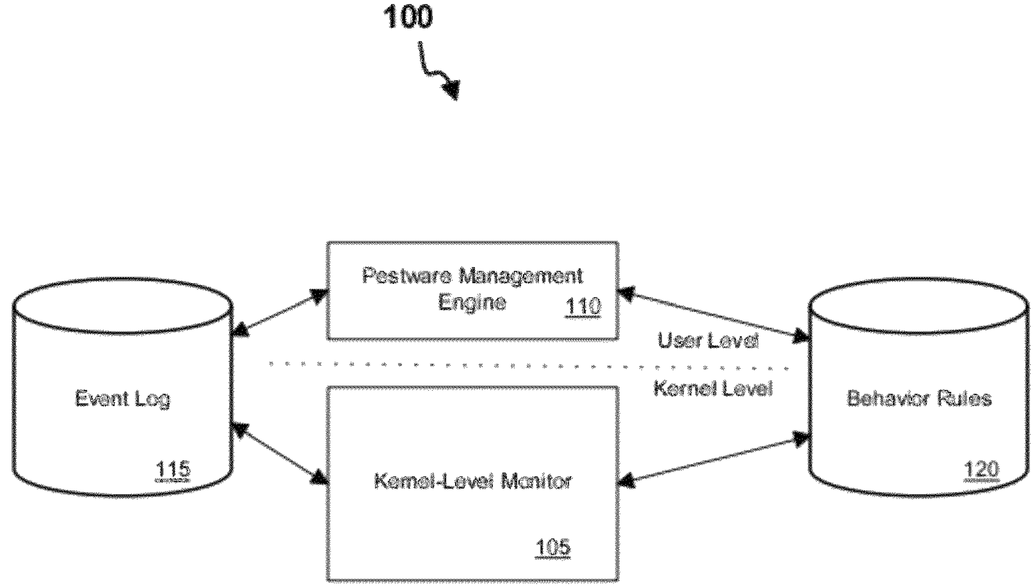
* cited by examiner

Primary Examiner — Madhuri Herzog
Assistant Examiner — Gregory Lane
(74) *Attorney, Agent, or Firm* — Merchant & Gould P.C.

(57) **ABSTRACT**

A system and method for managing pestware on a protected computer is described. The method in one variation includes monitoring events during a boot sequence of the computer; managing pestware-related events before native applications can run and after a kernel is loaded; managing pestware-related events when native applications can run; and scanning a registry of the computer for pestware when native applications can run. In variations, a pestware management engine is initialized after an operating system of the protected computer is initialized and the pestware management system both receives an event log of the monitored events and compiles the set of behavior rules utilized by kernel-level monitor.

20 Claims, 6 Drawing Sheets



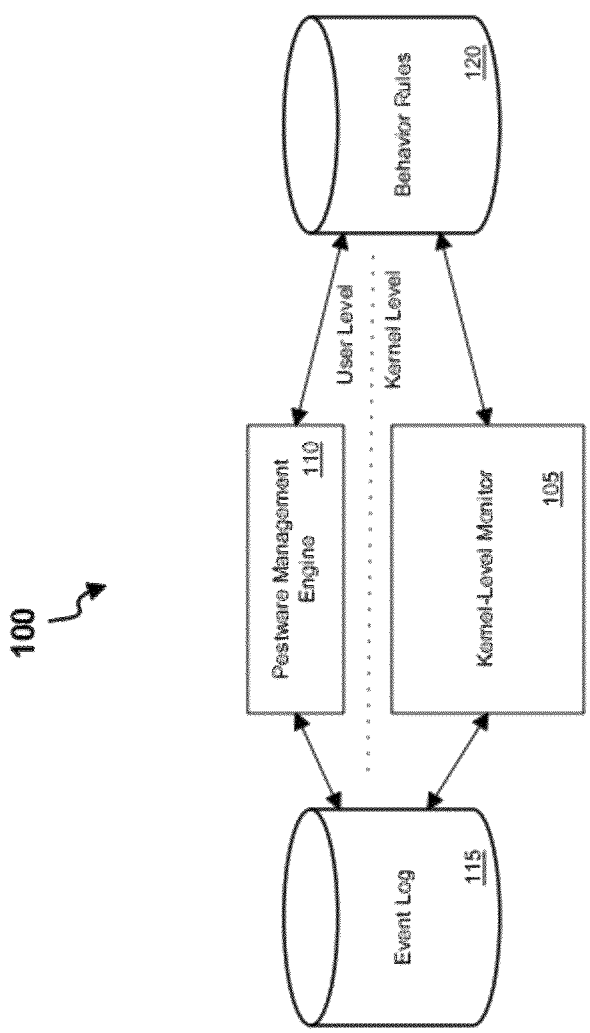

100
Behavior Rules
120
User Level
Kernel Level
Pestware Management Engine
110
Kernel-Level Monitor
105
Event Log
115

FIG. 1

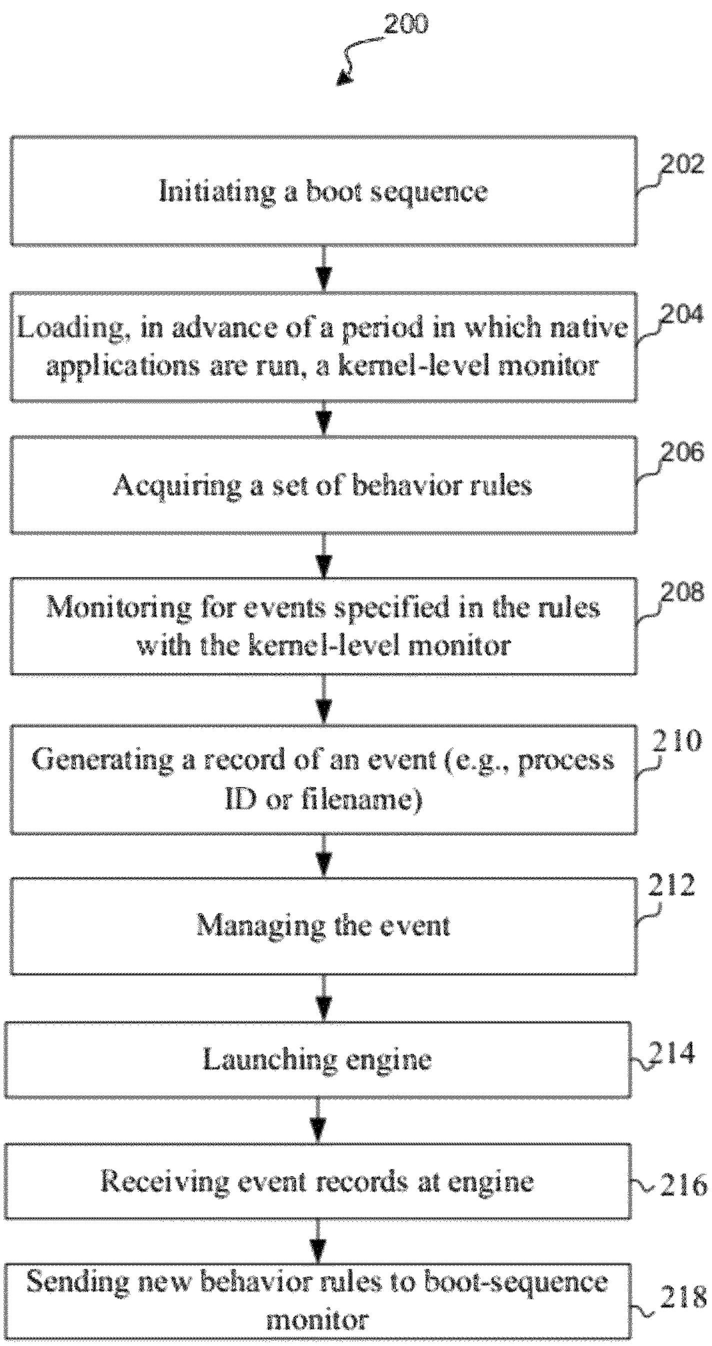
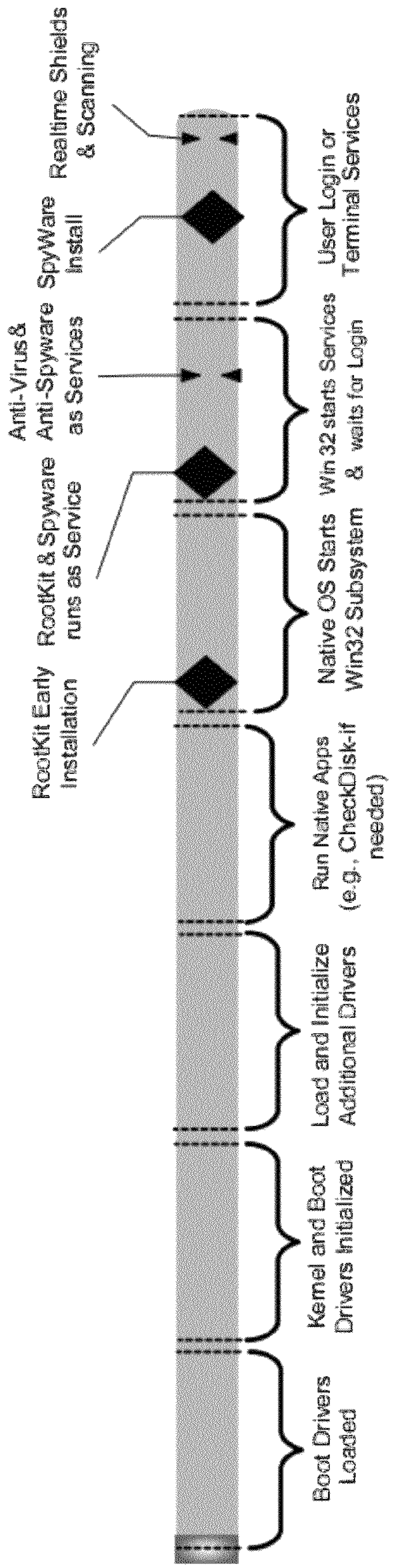

200
Initiating a boot sequence
202
Loading, in advance of a period in which native applications are run, a kernel-level monitor
204
Acquiring a set of behavior rules
206
Monitoring for events specified in the rules with the kernel-level monitor
208
Generating a record of an event (e.g., process ID or filename)
210
Managing the event
212
Launching engine
214
Receiving event records at engine
216
Sending new behavior rules to boot-sequence monitor
218

FIG. 2

Conventional Monitoring



(Prior Art)

FIG. 3

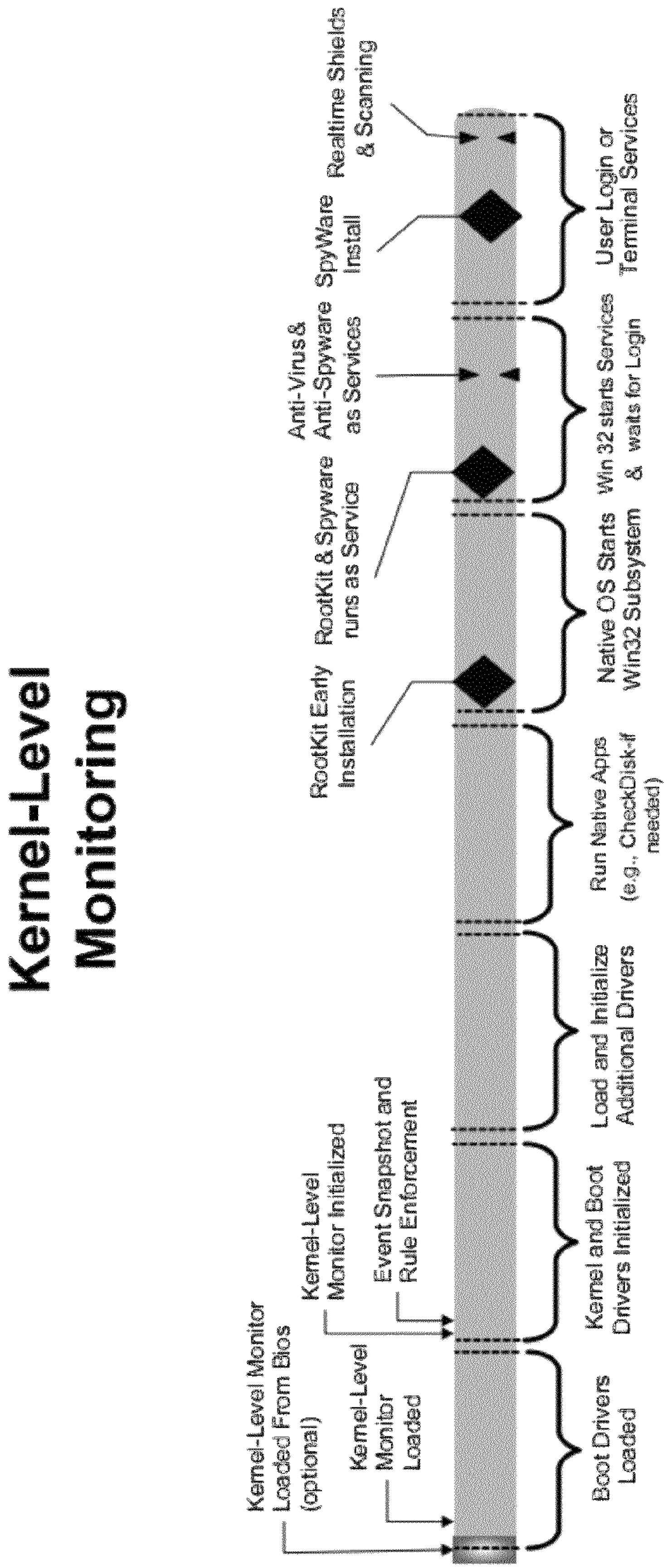

Kernel-Level Monitoring
Kernel-Level Monitor Loaded From Bios (optional)
Kernel-Level Monitor Loaded
Kernel-Level Monitor Initialized
Event Snapshot and Rule Enforcement
RootKit Early Installation
RootKit & Spyware runs as Service
Anti-Virus & Anti-Spyware as Services
SpyWare Install
Realtime Shields & Scanning
Boot Drivers Loaded
Kernel and Boot Drivers Initialized
Load and Initialize Additional Drivers
Run Native Apps (e.g., CheckDisk-if needed)
Native OS Starts Win32 Subsystem
Win 32 starts Services & waits for Login
User Login or Terminal Services

FIG. 4

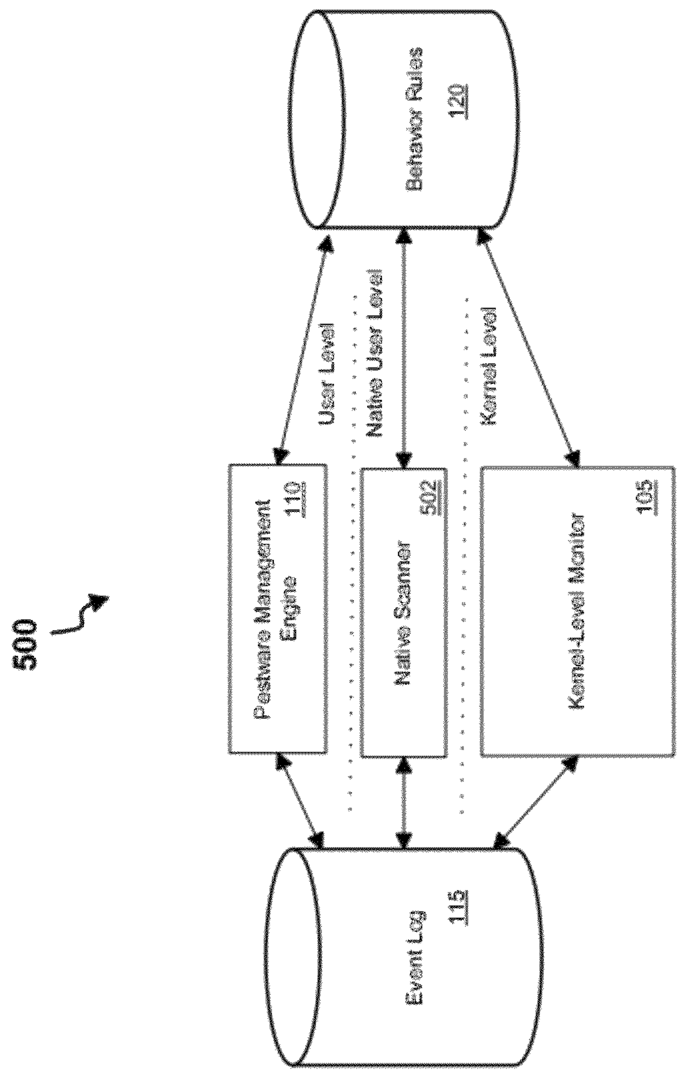

500
Behavior Rules
120
User Level
Native User Level
Kernel Level
Pestware Management Engine
110
Native Scanner
502
Kernel-Level Monitor
105
Event Log
115

FIG. 5

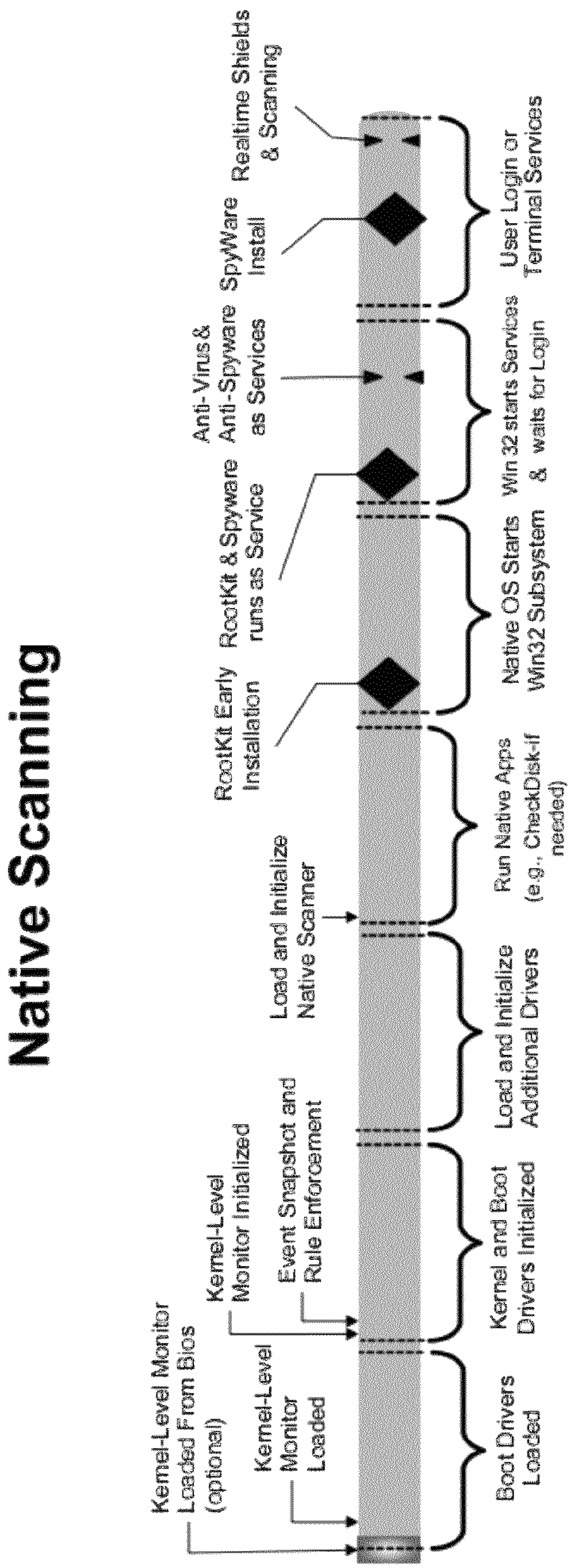

Native Scanning
Kernel-Level Monitor Loaded From Bios (optional)
Kernel-Level Monitor Loaded
Kernel-Level Monitor Initialized
Event Snapshot and Rule Enforcement
Load and Initialize Native Scanner
RootKit Early Installation
RootKit & Spyware runs as Service
Anti-Virus & Anti-Spyware as Services
SpyWare Install
Realtime Shields & Scanning
Boot Drivers Loaded
Kernel and Boot Drivers Initialized
Load and Initialize Additional Drivers
Run Native Apps (e.g., CheckDisk-if needed)
Native OS Starts Win32 Subsystem
Win 32 starts Services & waits for Login
User Login or Terminal Services

FIG. 6

MALWARE MANAGEMENT THROUGH KERNEL DETECTION DURING A BOOT SEQUENCE

PRIORITY

The present application is a continuation of commonly-owned and assigned U.S. Non-Provisioanl application Ser. No. 11/462,827, entitled "Malware Management Through Kernel Detection," filed Aug. 7, 2006, which is incorporated herein by reference in its entirety.

RELATED APPLICATIONS

The present application is also related to commonly-owned and assigned U.S. Non-Provisional application Ser. No. 10/956,578, entitled "System and Method for Monitoring Network Communications For Pestware," and U.S. Non-Provisional application Ser. No. 11/237,291, entitled "Client Side Exploit Tracking," each of which is incorporated herein by reference in its entirety.

FIELD OF THE INVENTION

The present invention relates to computer system management. In particular, but not by way of limitation, the present invention relates to systems and methods for controlling pestware or malware.

BACKGROUND OF THE INVENTION

Personal computers and business computers are continually attacked by trojans, spyware, and adware, collectively referred to as "malware" or "pestware." These types of programs generally act to gather information about a person or organization—often without the person or organization's knowledge. Some pestware is highly malicious. Other pestware is non-malicious but may cause issues with privacy or system performance. And yet other pestware is actual beneficial or wanted by the user. Wanted pestware is sometimes not characterized as "pestware" or "spyware." But, unless specified otherwise, "pestware" as used herein refers to any program that collects and/or reports information about a person or an organization and any "watcher processes" related to the pestware.

The design and implementation of current and future pestware incorporates techniques that make the pestware difficult to identify, remove, or even to detect. These techniques, and likely future improvements to them, rely on patches, hooks and yet-to-be-discovered methods for modifying the behavior of a computer operating system itself. Such techniques render current detection tools ineffective by intercepting and altering the results of operating system queries from the tools that must rely on the dependability of operating system calls to return lists of running programs, file system and registry contents, for example.

Detection of pestware that uses these cloaking techniques is often ineffective by the real-time shields that existing anti-pestware applications utilize because these real-time shields begin execution after pestware has been able to execute and modify the operating system. FIG. **3**, for example, illustrates the boot and operating system sequences that demonstrate the inability of user-mode services or applications to detect and prevent pestware (e.g., a Rootkit) that loads before them in a boot sequence. Periodic scanning for pestware is particularly ineffective because it leaves a window of time between scans in which pestware can execute and cloak itself. Accordingly, current software is not always able to identify and remove pestware in a convenient manner and will most certainly not be satisfactory in the future.

SUMMARY OF THE INVENTION

Exemplary embodiments of the present invention that are shown in the drawings are summarized below. These and other embodiments are more fully described in the Detailed Description section. It is to be understood, however, that there is no intention to limit the invention to the forms described in this Summary of the Invention or in the Detailed Description. One skilled in the art can recognize that there are numerous modifications, equivalents and alternative constructions that fall within the spirit and scope of the invention as expressed in the claims.

In one embodiment, the invention may be characterized as a method for managing pestware on a computer. The method in this embodiment includes monitoring events during a boot sequence of the computer; managing pestware-related events during a first period in a boot sequence of the computer, the first period in the boot sequence occuring before the computer becomes configured to run native applications, before a subsystem of an operating system is loaded, and after a kernel is loaded; managing pestware-related events in accordance with a set of behavior rules during a second period in the boot sequence occuring when the computer is configured to run native applications; and scanning a registry of the computer for pestware during the second period in the boot sequence.

As previously stated, the above-described embodiments and implementations are for illustration purposes only. Numerous other embodiments, implementations, and details of the invention are easily recognized by those of skill in the art from the following descriptions and claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Various objects and advantages and a more complete understanding of the present invention are apparent and more readily appreciated by reference to the following Detailed Description and to the appended claims when taken in conjunction with the accompanying Drawings wherein:

FIG. **1** is a block diagram depicting a pestware management system in accordance with an exemplary embodiment of the present invention;

FIG. **2** is a flowchart depicting steps carried out in accordance with an exemplary embodiment of the present invention;

FIG. **3** is a timing diagram depicting a boot sequence in accordance with prior art pestware management techniques;

FIG. **4** is a timing diagram depicting a boot sequence in accordance with pestware management techniques of several embodiments of the present invention;

FIG. **5** is a block diagram of a pestware management system in accordance with another embodiment of the present invention; and

FIG. **6** is a timing diagram depicting another boot sequence in accordance with another embodiment of the present invention.

DETAILED DESCRIPTION

Referring now to the drawings, where like or similar elements are designated with identical reference numerals throughout the several views, and referring in particular to FIG. **1**, it illustrates a block diagram **100** of one implementation of the present invention. This implementation includes

four components: a kernel-level monitor **105**, a pestware management engine **110**, event storage **115** and behavior rule storage **120**.

The kernel-level monitor **105** and a pestware management engine **110** can be implemented in software or hardware. And if implemented in software, the modules can be designed to operate on any type of computer system including WINDOWS and Linux-based systems Additionally, the software can be configured to operate on personal computers and/or servers. For convenience, embodiments of the present invention are generally described herein with relation to WINDOWS-based systems. Those of skill in the art can easily adapt these implementations for other types of operating systems or computer systems.

The event storage **115** and behavior rule storage **120** may be realized by a single magnetic hard drive, distributed hard drives or non-volatile memory. In some variations, the event storage **115** may be implemented with volatile memory.

Referring first to the kernel-level monitor **105**, in the exemplary embodiment it is responsible for detecting pestware or pestware activity on a protected computer or system. (The term "protected computer" is used to refer to any type of computer system, including personal computers, handheld computers, servers, firewalls, etc.) In several embodiments, the kernel-level monitor **105** begins execution early in the boot and operating system loading process on the protected computer, and as discussed further herein, the kernel-level monitor **105** obtains (e.g., in a secure manner), a most recent set of behavior rules from the behavior rules storage **120** that was previously compiled by the pestware management engine **110**.

The kernel-level monitor **105** additionally uses several techniques that allow it to intercept and monitor various operating system behaviors, including attempts by pestware to modify the behavior of the operating system through the use of patches and hooks or by invoking other mechanisms that could be used to alter information passed to or from the operating system.

Moreover, the kernel-level monitor **105** in the exemplary embodiment also provides mechanisms that can prevent, disable or disallow attempts by pestware to use or create patches, hooks and other methods required for intercepting or altering operating system information. In addition, the kernel-level monitor **105** identifies specific attempts by other software by locating the specific instance within the set of behavior rules **120** generated by the pestware management engine **110**.

Beneficially, the kernel-level monitor **105** in the exemplary embodiment also constructs and stores an event log in the event storage **115** that contains details for each of the intercepted and monitored events detected so that after the pestware management engine **110** executes, it acquires the event log from the event storage module **115**.

The pestware management engine **110** in the present embodiment identifies, by examining the event log created by the kernel-level monitor **105**, instances of known pestware, suspicious events by unknown software, and other patterns of events that may be useful in identifying pestware.

In addition, the pestware management engine **110** can add to or modify the common set of behavior rules that are utilized by kernel-level monitor **105** the next time the kernel-level monitor **105** loads and executes. In this way, the pestware management engine **110** is able to provide updated information about pestware-related events that should be prevented by the kernel-level monitor **105** during future boot operations on the protected computer.

Advantageously the ability to first observe, using the kernel-level monitor **105**, events associated with the loading, execution and behaviors of pestware, followed by the pestware management engines's **110** incorporation of the events into the behavior rules **120** allows, in subsequent boot operations, the kernel-level monitor **105** to prevent or deny the events from happening. Moreover, the pestware management module **110** may receive updated information about events that should be prevented from a remote host (not shown). The above-identified application entitled *Client-side Exploit Tracking*includes details about developing behavior rules at a centralized host location that may be pushed out to protected computers.

Referring next to FIG. **2**, it is a flowchart of one method for managing pestware. In this method, a boot sequence of the protected computer is initiated (e.g., by a user of the protected computer)(Block **202**), and in advance of a period when native applications (e.g., checkdisk) of the protected computer beginning execution, the kernel-level monitor **105** is initialized (Block **204**).

In several embodiments, for example, the kernel-level monitor **105** is loaded with boot drivers and initialized before at least some of the boot drivers. In some embodiments, the kernel-level monitor **105** is initialized before any of the boot drivers so as to be capable of monitoring the initialization of the boot drivers. In one embodiment, for example, a low tag number (e.g., the lowest tag number) is associated with the kernel-level monitor **105** so that the kernel-level monitor **105** is initialized before boot drivers, with higher tag numbers, are initialized. It is contemplated that once the kernel-level monitor **105** is provided with a low tag number, registry keys of the protected computer may be protected so as to prevent pestware from subsequently altering the tag order.

In yet another embodiment, a tag number of zero is associated with the kernel-level monitor **105** and the operating system of the protected computer is altered so as to allow a tag number of zero to be initialized first, before any boot drivers are initiated.

Referring briefly to FIG. **4**, which depicts an exemplary boot and operating system load sequence, the kernel-level monitor **105** in the exemplary embodiment is initialized before a period when native applications may be run. More specifically, as depicted in FIG. **4**, the kernel-level monitor **105** is loaded in the boot sequence when boot drivers are loaded and initialized at a very early (e.g., first) stage during the period in which the boot drivers are initialized. As depicted in FIG. **4**, in an alternative embodiment, the kernel-level monitor **105** is loaded from a BIOS of the protected computer.

In many embodiments of the invention, the kernel-level monitor **105** is realized by a Windows device driver that can operate in the constraints imposed on the boot drivers, which are needed to operate peripherals (e.g., keyboard, mouse, monitor, hard drive, etc.) during the initial boot process.

During the time when the kernel-level monitor **105** is loaded and initialized, driver hooks and monitoring mechanisms that are utilized by the kernel-level monitor **105** are placed into the operating system kernel. Referring again to FIG. **4**, this time period includes the time period depicted as "Event Snapshot and Rule Enforcement." In the exemplary boot sequence depicted in FIG. **4**, the kernel-level monitor **105** monitors events on an ongoing basis after being initialized early during the period in which the boot drivers are loaded.

Referring again to FIG. **2**, when the kernel-level monitor **105** begins execution, it acquires the most recent set of behavior rules (e.g., in a secure manner) from behavior rule storage **120** (Block **206**) and begins monitoring for the events specified in the rules (Block **208**). When a monitored event is

detected, details of the event (e.g., ProcessID and/or filename) are added to an event record (Block **210**) that is added to event storage **115**, which may be either an in-memory or a file-based log of events.

As shown in FIG. **2**, the kernel-level monitor **105** then manages the protected computer in response to the event in accordance with the behavior rules (Block **212**). For example, if a detected event is marked for denial or deletion in the behavior rules, the kernel-level monitor **105** either prevents the operating system from receiving the request or it forces the operating system to reject the request.

In the exemplary embodiment, as the boot sequence continues, the kernel-level monitor **105** continues to run independently, collecting event log data (e.g., during the periods in which additional drivers are initialized and when native applications are capable of executing). As depicted in FIG. **4**, as the boot sequence progresses, it is possible that a rootkit or another variety of pestware loads as a driver or service. In the context of a Windows operating system, for example, during the period that the native operating system starts the Win32 subsystem, it is possible that a rootkit or other pestware may load as a driver or service. Events that are generated by such a pestware driver or service are monitored by the kernel-level monitor **105**, added to the event log, and if the behavior rules include specific instance data indicating the event is a pestware-related event, the kernel-level monitor **105** denies or disables the rootkit's attempts to install.

As shown in FIG. **4**, in a Windows-based system, as Win32 starts global services, the kernel-level monitor **105** again monitors and possibly prevents spyware and rootkits from installing. Also during this period, the pestware management engine **110** is started as a service (Block **214**).

In several embodiments, one of the tasks of the pestware management engine **110** is to communicate securely with the kernel-level monitor **105** so as to retrieve the event logs for examination and analysis (Block **216**). In several embodiments, the event logs are compared against factors known to be associated with pestware so as to identify a likelihood that the events in the event log are pestware related events. In addition, the event log may be sent to a centralized host, which collects information about activities at protected computers and generates weighted factors which are pushed out to the protected computers and utilized to help identify pestware at the protected computer. The above-identified application entitled *Client Side Exploit Tracking* includes details of techniques that may be used to identify pestware-related events from the event logs.

In the exemplary embodiment, after the pestware management engine **110** is launched, it continues to communicate with the kernel-level monitor **105** so as to receive new events that are detected by the kernel-level monitor **105** and to send new behavior rules to the kernel-level monitor **105** as they are developed. If the pestware management engine **110** should cease execution, the kernel-level monitor **105** can continue to collect event logs and protect against those events covered by the current behavior rules until the pestware management engine **110** restarts and communicates once again with the kernel-level monitor **105**.

Referring next to FIG. **5**, shown a block diagram **500** of another implementation of the present invention. In this embodiment, a native scanner **502** is shown operating at a native user level of the protected computer. In several embodiments, the native scanner **502** is initialized early during the period of the boot sequence during which native applications may be run. In the embodiment depicted in FIG. **5**, for example, the native scanner **502** is the first non-kernel

application to initialize so as to be capable of monitoring the protected computer before any other native applications are initialized.

In operation, the native scanner **502** is configured to examine registry, filesystem and other portions of a protected computer before the Win32 subsystem and most services are loaded and executed. In this way, additional information is generated that the pestware management engine **110** can later examine and utilize in order to generate new behavior rules for the kernel-level monitor **105**.

In some embodiments, the native scanner **502** may be enabled with pestware management functions such as memory and file scanning The above-identified application entitled: *System and Method for Monitoring Network Communications* includes details of many pestware management functions that may be incorporated into the native scanner **502**.

In conclusion, the present invention provides, among other things, a system and method for managing pestware during a boot sequence of a protected computer. It should be recognized, however, that embodiments of the present invention certainly have applications that extend beyond the boot sequence period of operation when the protected computer is operating in a user mode. Moreover, those skilled in the art can readily recognize that numerous variations and substitutions may be made in the invention, its use and its configuration to achieve substantially the same results as achieved by the embodiments described herein. Accordingly, there is no intention to limit the invention to the disclosed exemplary forms. Many variations, modifications and alternative constructions fall within the scope and spirit of the disclosed invention as expressed in the claims.

What is claimed is:

1. A method for managing pestware on a computer comprising:

monitoring events during a boot sequence of the computer;

managing pestware-related events during a first period in a boot sequence of the computer, the first period in the boot sequence occurring before the computer becomes configured to run native applications, before a subsystem of an operating system is loaded, and after a kernel is loaded;

managing pestware-related events in accordance with a set of behavior rules during a second period in the boot sequence occurring when the computer is configured to run native applications;

generating, in response to the monitoring, a record of events, the record of events including the pestware-related events;

analyzing the record of events so as to identify the pestware-related events

modifying the set of behavior rules so as to prevent the pestware related events; and

scanning data in a registry of the computer for pestware during the second period in the boot sequence.

2. The method of claim **1**, further comprising launching, after an operating system is initiated, a pestware management engine; and

wherein the set of behavior rules includes behavior rules compiled by the pestware management engine.

3. The method of claim **1**, wherein the record of events includes information selected from a group consisting of: process identification information, file identification information, and hook generation information.

4. The method of claim **1**, further comprising:

managing pestware-related events after the computer becomes configured to run native applications.

5. The method of claim **1**, wherein the monitoring includes monitoring the boot sequence while boot drivers are initiated.

6. A system for managing pestware on a computer comprising:

a processor;

a memory comprising:

a first instruction set configured to monitor events during a boot sequence of the computer;

a second instruction set configured to manage pestware-related events during first period in a boot sequence of the computer, the first period in the boot sequence occurring before the computer becomes configured to run native applications, before a WIN32 subsystem is loaded, and after a kernel is loaded;

a third instruction set configured to manage pestware-related events in accordance with a set of behavior rules during a second period in the boot sequence occurring when the computer is configured to run native applications;

a fourth instruction set configured to generate, in response to the monitoring, a record of events, the record of events including the pestware-related events;

a fifth instruction set configured to analyze the record of events so as to identify the pestware-related events; and

a sixth instruction set configured to modify the set of behavior rules so as to prevent the pestware related events; and

a seventh instruction set configured to scan data in a registry of the computer for pestware during the second period in the boot sequence.

7. The system of claim **6**, wherein the memory further comprises an eighth instruction set configured to launch, after an operating system is initiated, a pestware management engine; and wherein the set of behavior rules includes behavior rules compiled by the pestware management engine.

8. The system of claim **1**, wherein the record of events includes information selected from a group consisting of: process identification information, file identification information, and hook generation information.

9. The system of claim **6**, wherein the memory further comprises:

a ninth instruction set configured to manage pestware-related events after the computer becomes configured to run native applications.

10. The system of claim **6**, wherein the first instruction set is further configured to monitor the boot sequence while boot drivers are initiated.

11. A non-transitory tangible computer-readable storage medium comprising program instructions for:

monitoring events during a boot sequence of a computer;

managing pestware-related events during a first period in a boot sequence of the computer, the first period in the boot sequence occurring before the computer becomes configured to run native applications, before a WIN32 subsystem is loaded, and after a kernel is loaded;

managing pestware-related events in accordance with a set of behavior rules during a second period in the boot sequence occurring when the computer is configured to run native applications;

generating, in response to the monitoring, a record of events, the record of events including the pestware-related events;

analyzing the record of events so as to identify the pestware-related events;

modifying the set of behavior rules so as to prevent the pestware related events; and

scanning data in a registry of the computer for pestware during the second period in the boot sequence.

12. The non-transitory tangible computer-readable storage medium of claim **11** further comprising program instructions for launching, after an operating system is initiated, a pestware management engine; and

wherein the set of behavior rules includes behavior rules compiled by the pestware management engine.

13. The non-transitory tangible computer-readable storage medium of claim **11**, wherein the record of events includes information selected from a group consisting of: process identification information, file identification information, and hook generation information.

14. The non-transitory tangible computer-readable storage medium of claim **11** further comprising program instructions for:

managing pestware-related events after the computer becomes configured to run native applications.

15. The non-transitory tangible computer-readable storage medium of claim **11**, wherein the monitoring includes monitoring the boot sequence while boot drivers are initiated.

16. The method of claim **2**, further comprising modifying, by the pestware management engine, the behavior rules utilized by a kernel-level monitor, wherein the kernel-level monitor manages pestware-related events in the first and second periods in the boot sequence.

17. The method of claim **1**, further comprising modifying the set of behavior rules based on the scanned registry so as to prevent the pestware-related events.

18. The method of claim **1**, wherein the pestware-related invents include events associated with at least one of loading pestware, executing pestware, and one or more behaviors of pestware.

19. The system of claim **6**, wherein the pestware-related invents include events associated with at least one of loading pestware, executing pestware, and one or more behaviors of pestware.

20. The non-transitory tangible computer-readable storage medium of claim **11**, wherein the pestware-related invents include events associated with at least one of loading pestware, executing pestware, and one or more behaviors of pestware.

* * * * *

Exhibit 12

US008719932B2

(12) **United States Patent**
Boney

(10) **Patent No.: US 8,719,932 B2**
(45) **Date of Patent: *May 6, 2014**

(54) **BACKWARDS RESEARCHING ACTIVITY INDICATIVE OF PESTWARE**

(75) Inventor: **Matthew L. Boney**, Longmont, CO (US)

(73) Assignee: **Webroot Inc.**, Broomfield, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 142 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **13/490,294**

(22) Filed: **Jun. 6, 2012**

(65) **Prior Publication Data**

US 2012/0246722 A1 Sep. 27, 2012

Related U.S. Application Data

(63) Continuation of application No. 11/408,146, filed on Apr. 20, 2006, now Pat. No. 8,201,243.

(51) **Int. Cl.**
G06F 21/56 (2013.01)

(52) **U.S. Cl.**
USPC **726/22**; 713/164; 713/202; 713/188; 709/206; 709/224; 380/279; 370/386; 726/27

(58) **Field of Classification Search**
CPC H04L 63/1416; G06F 21/56
USPC 726/22
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,490,352	B2 *	2/2009	Kramer et al.	726/22
8,312,479	B2 *	11/2012	Boillot	719/328
2005/0273858	A1 *	12/2005	Zadok et al.	726/24
2007/0016914	A1 *	1/2007	Yeap	719/328
2007/0174911	A1 *	7/2007	Kronenberg et al.	726/22
2011/0167050	A1 *	7/2011	Fanton et al.	707/698

OTHER PUBLICATIONS

The ethical and legal concerns of spyware|http://utopia.csis.pace.edu/dps/2007/jkile/2005%20-%20Spring/DCS823/Spyware/sipior.pdf|2005|Sipior et al.|pp. 39-49.*

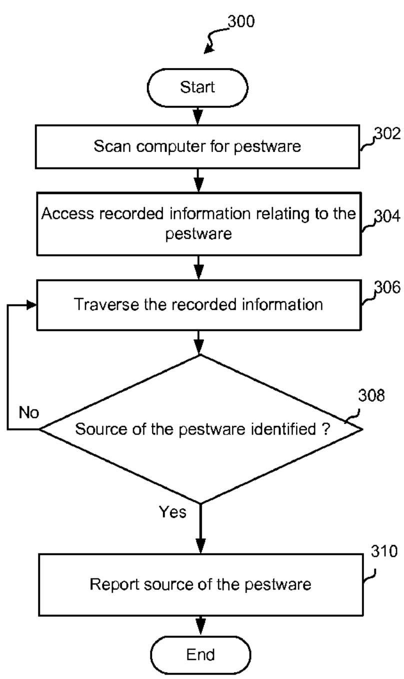
* cited by examiner

Primary Examiner — Mahfuzur Rahman
(74) *Attorney, Agent, or Firm* — Sheridan Ross P.C.

(57) **ABSTRACT**

A system and method for researching an identity of a source of activity that is indicative of pestware is described. In one embodiment the method comprises monitoring, using a kernel-mode driver, API call activity on the computer; storing information related to the API call activity in a log; analyzing, heuristically, the API call activity to determine whether one or more weighted factors associated with the API call activity exceeds a threshold; identifying, based upon the API call activity, a suspected pestware object on the computer; identifying, in response to the identifying the suspected pestware object, a reference to an identity of an externally networked source of the suspected pestware object; and reporting the identity of the externally networked source to an externally networked pestware research entity.

15 Claims, 5 Drawing Sheets



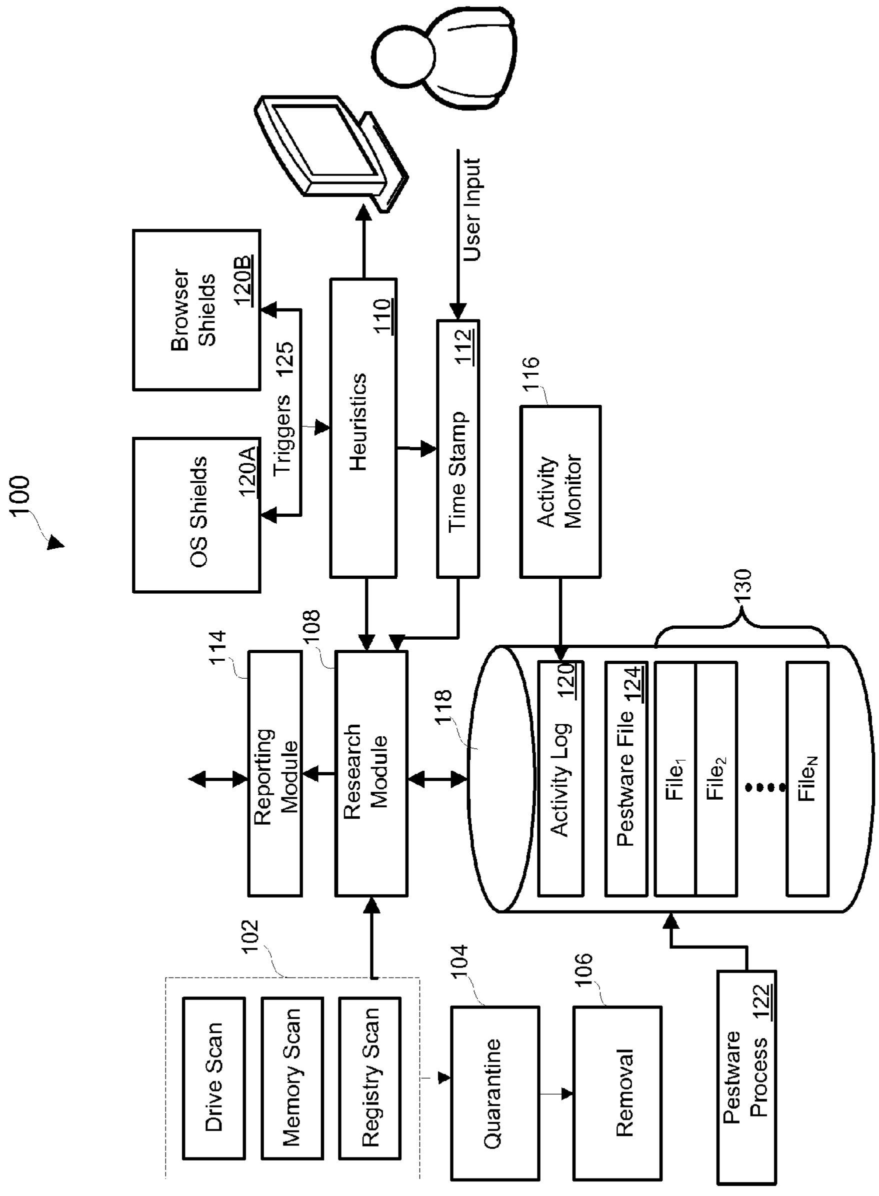

100
OS Shields 120A
Browser Shields 120B
Triggers 125
Heuristics 110
Time Stamp 112
User Input
116
Activity Monitor
114
Reporting Module
108
Research Module
118
Activity Log 120
Pestware File 124
File1
File2
FileN
130
102
Drive Scan
Memory Scan
Registry Scan
104
Quarantine
106
Removal
Pestware Process 122

FIGURE 1

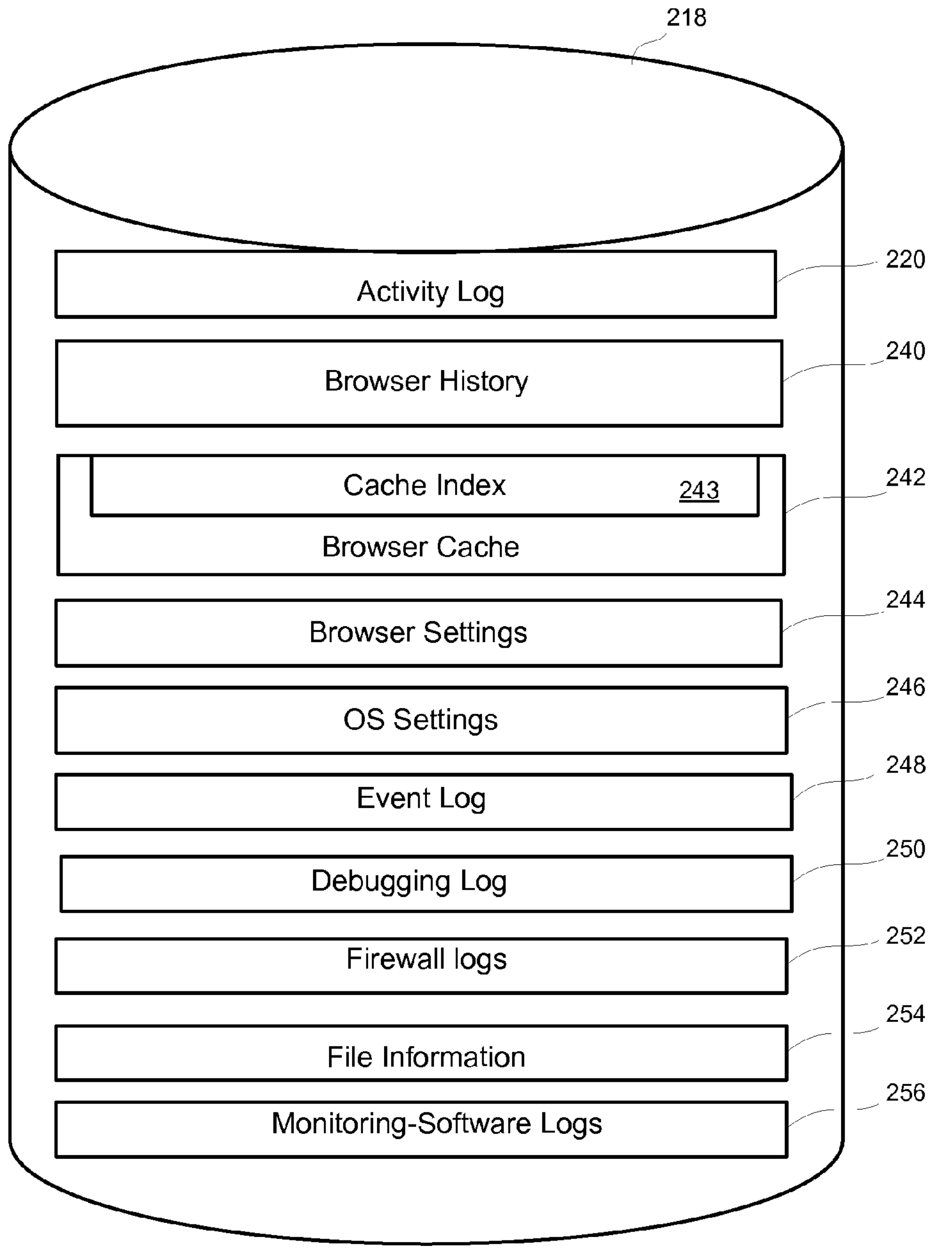
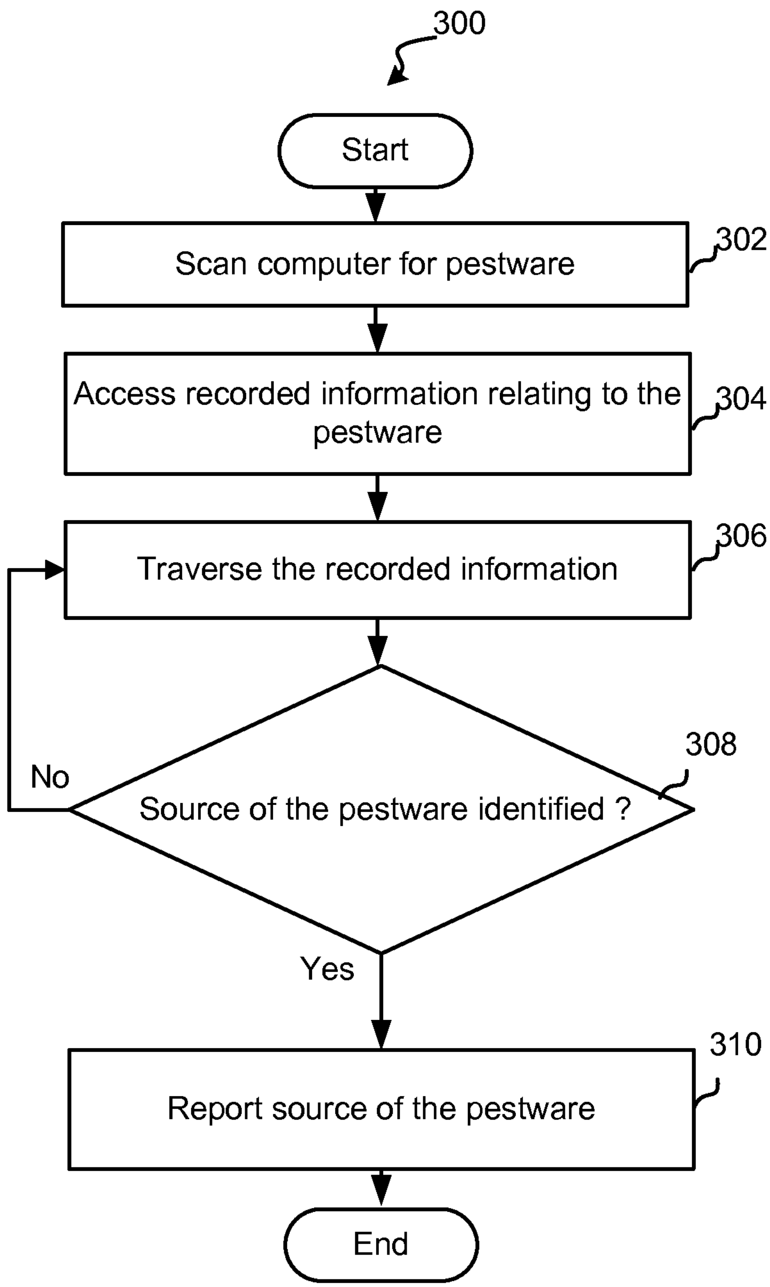

218
Activity Log
220
Browser History
240
Cache Index
243
Browser Cache
242
Browser Settings
244
OS Settings
246
Event Log
248
Debugging Log
250
Firewall logs
252
File Information
254
Monitoring-Software Logs
256

FIGURE 2



300
Start
Scan computer for pestware
302
Access recorded information relating to the pestware
304
Traverse the recorded information
306
308
No
Source of the pestware identified ?
Yes
Report source of the pestware
310
End

FIGURE 3

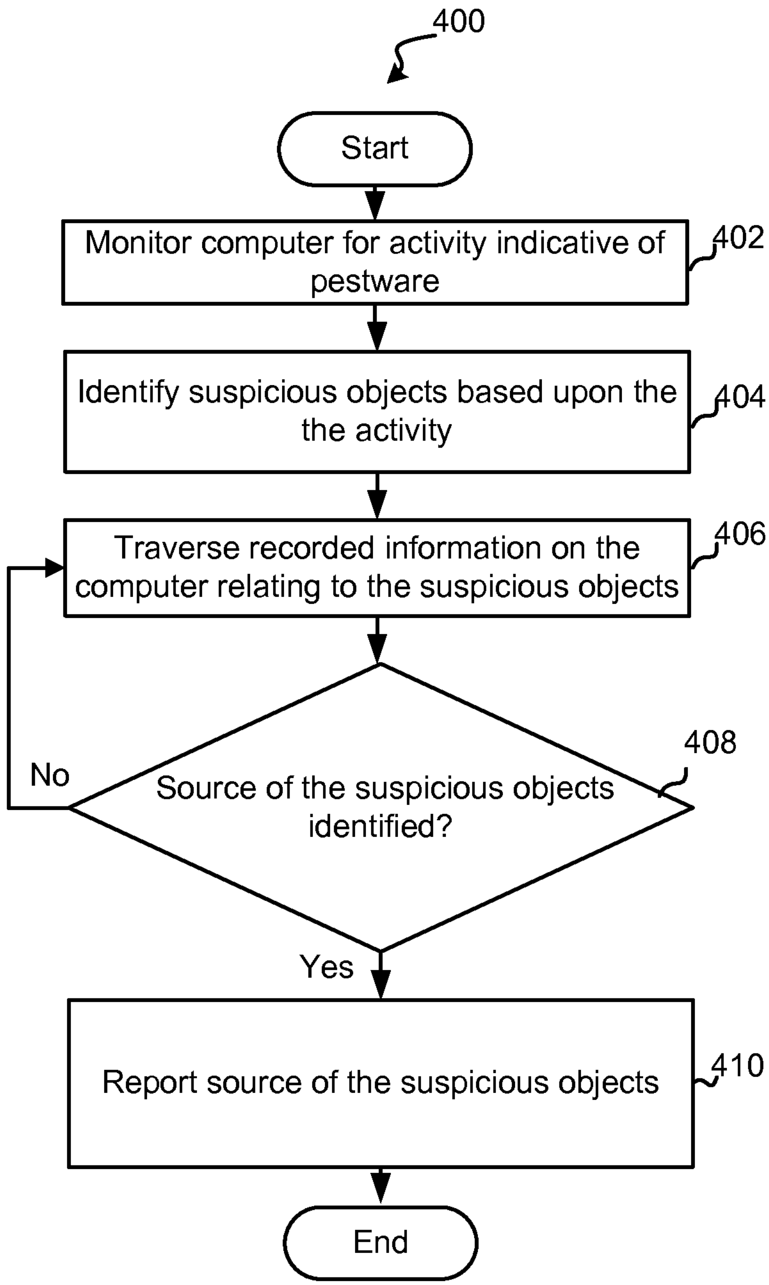

400
Start
Monitor computer for activity indicative of pestware
402
Identify suspicious objects based upon the the activity
404
Traverse recorded information on the computer relating to the suspicious objects
406
Source of the suspicious objects identified?
408
No
Yes
Report source of the suspicious objects
410
End

FIGURE 4

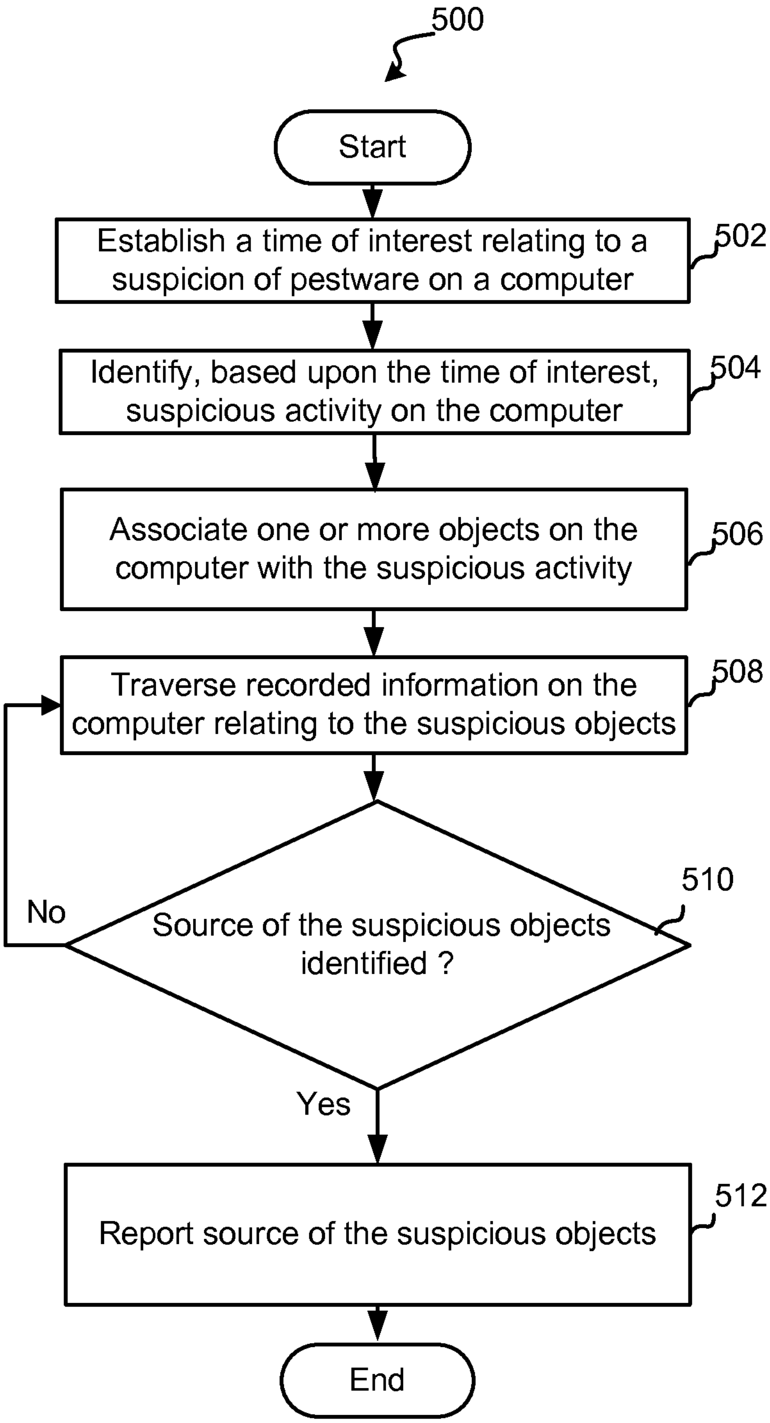

500
Start
Establish a time of interest relating to a suspicion of pestware on a computer
502
Identify, based upon the time of interest, suspicious activity on the computer
504
Associate one or more objects on the computer with the suspicious activity
506
Traverse recorded information on the computer relating to the suspicious objects
508
No
Source of the suspicious objects identified ?
510
Yes
Report source of the suspicious objects
512
End

FIGURE 5

BACKWARDS RESEARCHING ACTIVITY INDICATIVE OF PESTWARE

PRIORITY

The present application is a continuation of commonly-owned and assigned application Ser. No. 11/408,146, entitled Backwards Researching Activity Indicative of Pestware, filed Apr. 20, 2006, which is incorporated herein by reference in its entirety and for all purposes.

RELATED APPLICATIONS

The present application is related to commonly-owned and assigned application Ser. No. 10/956,573, entitled System and Method For Heuristic Analysis to Identify Pestware, which issued as U.S. Pat. No. 7,480,683; application Ser. No. 10/956,574, entitled System and Method for Pestware Detection and Removal which issued as U.S. Pat. No. 7,533,131; application Ser. No. 11/237,291, entitled System and Method for Locating Malware and Generating Malware Definitions; application Ser. No. 11/257,609, entitled System and Method for Kernel-Level Pestware Management; application Ser. No. 11/237,291, entitled Client Side Exploit Tracking; application Ser. No. 11/408,215, entitled Backward Researching Existing Pestware; and application Ser. No. 11/408,145, entitled Backward Researching Time Stamped Events to Find an Origin of Pestware, which are incorporated herein by reference.

COPYRIGHT

A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

FIELD OF THE INVENTION

The present invention relates to computer system management. In particular, but not by way of limitation, the present invention relates to systems and methods for detecting, controlling and/or removing pestware.

BACKGROUND OF THE INVENTION

Personal computers and business computers are continually attacked by trojans, spyware, and adware, collectively referred to as “malware,” “spyware” or “pestware.” These types of programs generally act to gather information about a person or organization—often without the person or organization’s knowledge. Some pestware is highly malicious. Other pestware is non-malicious but may cause issues with privacy or system performance. And yet other pestware is actually beneficial or wanted by the user. Unless specified otherwise, “pestware” as used herein refers to any of these programs that collects information about a person or an organization.

Software is presently available to detect and remove pestware. But as it evolves, the software to detect and remove it must also evolve. Accordingly, current techniques and software for removing pestware are not always satisfactory and will most certainly not be satisfactory in the future. Additionally, because some pestware is actually valuable to a user, pestware-detection software should, in some cases, be able to handle differences between wanted and unwanted pestware.

Current pestware removal software uses definitions of known pestware to search for and remove files on a protected system. These definitions are often slow and cumbersome to create. Additionally, it is often difficult to initially locate the pestware in order to create the definitions. Accordingly, a system and method are needed to address the shortfalls of present technology and to provide other new and innovative features.

SUMMARY OF THE INVENTION

Exemplary embodiments of the present invention that are shown in the drawings are summarized below. These and other embodiments are more fully described in the Detailed Description section. It is to be understood, however, that there is no intention to limit the invention to the forms described in this Summary of the Invention or in the Detailed Description. One skilled in the art can recognize that there are numerous modifications, equivalents and alternative constructions that fall within the spirit and scope of the invention as expressed in the claims.

The present invention can provide a system and method for researching an identity of a source of activity that is indicative of pestware. In one embodiment the method comprises monitoring, using a kernel-mode driver, API call activity on the computer; storing information related to the API call activity in a log; analyzing, heuristically, the API call activity to determine whether one or more weighted factors associated with the API call activity exceeds a threshold; identifying, based upon the API call activity, a suspected pestware object on the computer; identifying, in response to the identifying the suspected pestware object, a reference to an identity of an externally networked source of the suspected pestware object; and reporting the identity of the externally networked source to an externally networked pestware research entity.

As previously stated, the above-described embodiments and implementations are for illustration purposes only. Numerous other embodiments, implementations, and details of the invention are easily recognized by those of skill in the art from the following descriptions and claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Various objects and advantages and a more complete understanding of the present invention are apparent and more readily appreciated by reference to the following Detailed Description and to the appended claims when taken in conjunction with the accompanying Drawings wherein:

FIG. **1** illustrates a block diagram of one implementation of the present invention;

FIG. **2** is a block diagram depicting an embodiment of the file storage device of FIG. **1**;

FIG. **3** is a flowchart depicting a method according to an exemplary embodiment;

FIG. **4** is a flowchart depicting another method according to another embodiment;

FIG. **5** is a flowchart depicting yet another method according to yet another embodiment.

DETAILED DESCRIPTION

Referring now to the drawings, where like or similar elements are designated with identical reference numerals throughout the several views, and referring in particular to FIG. **1**, it illustrates a block diagram of one implementation of

the present invention. Shown is a protected computer **100** that includes a detection module **102**, quarantine module **104**, removal module **106**, and shields **120**. In addition, a research module **108** is shown coupled to a heuristics module **110**, a time stamp module **112** and a reporting module **114**.

Each of these modules can be implemented in software or hardware. And if implemented in software, the modules can be implemented in a single software package or in multiple software packages. In addition, one of ordinary skill in the art will recognize that the software can be designed to operate on any type of computer system including WINDOWS and Linux-based systems. Additionally, the software can be configured to operate on personal computers and/or servers. For convenience, embodiments of the present invention are generally described herein with relation to WINDOWS-based systems. Those of skill in the art can easily adapt these implementations for other types of operating systems or computer systems.

Also shown is a file storage device **118** that is coupled to the research module **108** and an activity monitor **116**. In this embodiment the file storage device includes an activity log **120**, a pestware file **124** and a collection of N files **130**. The file storage device **118** is described herein in several implementations as hard disk drive for convenience, but this is certainly not required, and one of ordinary skill in the art will recognize that other storage media may be utilized without departing from the scope of the present invention. In addition, one of ordinary skill in the art will recognize that the storage device **118**, which is depicted for convenience as a single storage device, may be realized by multiple (e.g., distributed) storage devices.

In the exemplary embodiment, the pestware file **124**, corresponds to (e.g., includes data relating to) a pestware process **122** operating in memory. The pestware process **122** is exemplary of pestware processes that are configured to make one or more unauthorized alterations to the computer **100**. For example, the pestware process **122** may make changes to either or both of the browser settings and/or operating system (OS) settings without approval and/or the knowledge of the user.

In accordance with several embodiments, the research module **108** is configured to receive an indication that either known pestware is residing on the computer or activities indicative of pestware have occurred or are occurring on the protected computer. In response, the research module **108** is configured to research the activity log **120** and/or the N files **130**, which include information (e.g., historical logs) relating to events on the computer **100**, to identify a source of the pestware **122**, **124** or pestware-related activities, which may be reported by the reporting module **114** to a centralized data store for subsequent pestware management purposes.

For example, the identities (e.g., I.P. address, URL, email client or program name) of sources (e.g., web sites, email or program) of the pestware may be collected in a centralized data store (not shown) and then subsequently reported to other users. In addition, the sources of the pestware or suspected pestware may be visited to further research how the pestware is being distributed from the URLs and/or to generate new definitions for pestware discovered at these sources.

As described further with reference to FIG. **2**, the N files **130** include one or more files with information that assist the research module **108** in tracing information in the N files **130** to an identity (e.g., URL) of the source of pestware **122**, **124** on the computer **100**. One or more of the N files **130** may be associated with an operating system of the protected computer and/or one or more applications of the protected computer, and may include information such as process IDs,

registry entries, file names, cookies and URLs among other information that is used to trace from an identified pestware file, pestware process and/or pestware activity to an originating source (e.g., URL or IP address) of the infection. In one embodiment, one or more of the N files **130** is generated from an application that generates a log of data after examining the computer. An example of such an application is an application distributed under the name HijackThis.

In the exemplary embodiment depicted in FIG. **1**, the research module **108** is configured to receive indications that pestware may be present on the computer from each of the detection engine **102**, the heuristics engine **110** and the time stamp module **112**, but this is certainly not required. In other embodiments, for example, the research module **108** may be configured to communicate only with the detection engine **102** or only with the heuristics engine. In yet other embodiments, the research module **108** may be configured to receive only a time stamp from the time stamp module **112**.

According to several embodiments, pestware-detection functions operating on the protected computer **100** are represented by the detection engine **102**, the quarantine engine **104**, the removal engine **106**, the shields **120** and the heuristic engine **110**. The basic functions of the detection engine **102** is to compare files, processes and registry entries on the protected computer against known pestware definitions and characteristics. When a match is found, in addition to quarantining and removing a pestware object, the detection engine **102** informs the research module **108** of the pestware. Details associated with several embodiments of sweep, quarantine, and removal engines are found in the above-identified application entitled System and Method for Pestware Detection and Removal.

Pestware and pestware activity can also be identified by the shields **120**, which generally run in the background on the computer system. In the exemplary embodiment depicted in FIG. **1**, the shields **120** are divided into the operating system shields **120**A and the browser shields **120**B. The shields **120** are designed to watch for pestware and for typical pestware activity and includes two types of shields: behavior-monitoring shields and definition-based shields.

As an example, the shields **120** monitor the protected computer **100** for certain types of activities that generally correspond to pestware behavior. Particular examples of some of the types of activities that are monitored include a process spawning another process, an alteration to registry entries, communications with remote sites via the Internet, alterations to a start up folder, injection of a DLL into another process, and a change to the browser's home page and/or bookmarks. Some specific examples of shields are disclosed in the above-identified application entitled System and Method for Locating Malware and Generating Malware Definitions. In the exemplary embodiment, the shields **120** inform the heuristics engine **110** about the activities and the heuristics engine **110** determines whether the research module **108** should be informed and/or whether the activity should be blocked.

As an example, the heuristics module **110** may compare the activities on the computer with weighted factors to make decisions relative to activities at the protected computer. Each of these factors may be, for example, associated with a particular activity and each factor may be weighted by the likelihood that the activity is associated with pestware. If the sum of the weighted factors that match the activity history exceed a threshold, then the activity is identified as pestware activity and the heuristics module **110** prompts the research module **108** to initiate research into the origin of the pestware initiating the activity. It should be recognized that this type of heuristics operation is merely one example, and that the heu-

ristics module **110** may use other techniques to analyze activity on the computer. Additional information related to heuristics-based scanning is found in the above-identified applications entitled System and Method For Heuristic Analysis to Identify Pestware and Client Side Exploit Tracking.

In the exemplary embodiment depicted in FIG. **1**, the time stamp module **112** is configured to send a time stamp to the research module **108** in response to a request from a user and/or in response to the heuristics module **110**. In some embodiments for example, the heuristics module **110** provides the user with information about suspect activity on the computer **100** so that the user has the option as to whether or not the research module **108** will attempt to identify the source of the activity.

In other embodiments, the heuristics module **110** prompts the time stamp module **112** to initiate the generation of a time stamp, without user intervention, in response to activity that is indicative of pestware activity. In one implementation, for example, the heuristics module **110** prompts the research module **108** to initiate tracing of the pestware activity to an origin of the pestware that is associated with the activity, and the heuristics module **110** also prompts the time stamp module **112** to send a time stamp to the research module **108** so that the research module **108** is provided with a time reference as well as information about the pestware activities.

In yet other embodiments, the timestamp module **112** operates independently of the heuristics module **110**. For example, the timestamp module **112** may prompt the research module **108** to initiate a review of the activity log **120** and/or one or more of the N files in response to a user request.

The activity monitor **116** in several embodiments monitors activities on the computer and stores information about the activities in the activity log **120** so as to assist the research module **108** in identifying activities associated with pestware and the source of the pestware. In some embodiments, for example, the activity log **120** includes a list of processes that are running on the protected computer and the files that are associated with the processes. Although not depicted in FIG. **1**, the shields **120** may utilize the activity monitor **116** to detect pestware-related events and intercept efforts by pestware to spawn new processes or alter aspects of the protected computer **100**.

In some variations, the activity monitor **116** may inject code into existing processes so that when a process attempts to call a function of the computer's **100** operating system (not shown) the injected code can check the process to be started and raise a flag if the existing process is attempting to create a new pestware process or attempting to alter one or more settings (e.g., a registry setting) of the protected computer **100**.

In other embodiments, the activity monitor **116** is realized by a kernel-mode driver that may be loaded during a boot sequence for the protected computer or anytime later. In these embodiments, the activity monitor **116** is configured to log API calls in the activity log. In many variations for example, when a process (e.g., the pestware process) attempts to spawn a another pestware process or alter a registry entry, the API call utilized by the process is intercepted before it is carried out by an operating system of the protected computer. In this way, the attempt may be logged in the activity log **120** and the process may be analyzed to determine whether it is known to be a pestware-related process. Additional details of use of a kernel-level driver in connection with pestware management may be found in the above identified application entitled: System and Method for Kernel-Level Pestware Management.

It should be recognized that the block diagram in FIG. **1** depicts functional capabilities associated with several embodiments of the present invention. One of ordinary skill in the art will recognize that the functions described with reference to FIG. **1** may be realized by various implementations of software in connection with hardware or hardware alone. In these implementations several functions may be consolidated into a single module, for example, and as a consequence, may appear different from the block diagram in FIG. **1** without departing from the scope of the present invention.

Referring next to FIG. **2**, shown are exemplary resources, also referred to herein as historical logs, that are available to the research module **108** in accordance with one embodiment of the present invention. In general, these resources are logs of historical events that occurred on the computer, and each of the logs includes information that may be used to reference other information when, for example, activities, processes and files are traced so as to determine an origin of the pestware or suspected pestware. As shown, a file storage device **218** in this embodiment includes an activity log **220**, which may be generated by an activity monitor (e.g., the activity monitor **116**) and may include information about processes running on the computer **100** and files corresponding to the processes. In variations, the activity log **220** includes a history of API calls and respective times made by processes running on the protected computer, but this is certainly not required.

In addition, the file storage device **218** includes a browser history **240**, browser cache **242**, browser settings **244**, OS settings **246**, an event log **248**, a debugging log **250**, a firewall log **252**, file information **254** and monitoring software logs **256**. One or more of these exemplary files **240-256** may be implemented for one or more of the N files **130** described with reference to FIG. **1**.

The browser history **240** in this embodiment includes a listing of web sites and respective times that the web sites were visited by the user. The browser cache **242** in this embodiment includes files, cookies, and other objects cached in connection with use of a browser of the protected computer (e.g., Internet Explorer or Mozilla (not shown)). As depicted, the browser cache **242** includes a cache index **243** that includes a listing, which associates the content of the browser cache **242** with URLs and time stamps. As discussed further herein, the cache index **243** provides an efficient means for identifying the times objects were retrieved and the URLs that the objects were retrieved from.

The browser settings **244** include information about settings associated with a browser and may include a home page setting and list of user favorites. The browser settings **244** are monitored by the shields **120** for changes. Those settings also contain URLs that may be referenced in time stamped logs, firewall logs, browser histories, etc.

The operating system (OS) settings **246** may include registry entries, a start up folder and other information utilized by an operating system of the protected computer. As discussed further herein, data included in the OS settings **246** may include time stamps, which indicate when changes were made to the settings.

The event log **248** in this embodiment includes a log of events that is maintained by an operating system of the protected computer **100**. For example, the event log **248** may include event information including errors, user log-in history, a listing of processes that have been launched (and the users that launched the processes), path information, information about which process (and which users) accessed a secure area and other information relating to operations of the computer **100**.

Also shown in FIG. **2** is a debugging log **250** that includes application errors, which point to a process and the address where the error occurred. Some techniques employed by pestware forcibly shut down applications, or cause applications to crash when their memory space is injected with pestware code. The infected applications like "explorer.exe" will crash, or some times restart spontaneously. These events/occurrences show up in debugging logs. These are time stamped events, that also reference files on the system.

The firewall log **252** is this embodiment is a collection of information relating to network-related events on the protected computer. The firewall log **252**, may for example, include time stamps of network activities on the protected computer, which may be utilized by the research module **108** to locate pestware or indicia of pestware.

Also shown in the data storage device **218** is a collection of file information **254**, also known as the file system, which includes a database that the operating system uses to locate and store information about files. For example, the file information **254** may include the date and time a file is created, a date and time the file was last modified, the date and time the files was last accessed and the size of the file.

The monitoring-software logs **256** includes information collected from one or more pieces of monitoring software that are configured to monitor activity on the computer. As an example, the monitoring logs **256** may be generated from a filter driver, a module interfacing with a layer service provider and/or browser helper objects. It should be recognized that the logs **220**, **240-256** depicted in FIG. **2** are merely exemplary and these logs are by no means representative all the potential logs that may be accessed to research origins of pestware and/or suspected-pestware.

Referring next to FIG. **3**, shown is a flowchart depicting a method in accordance with one embodiment. Although reference will be made to FIGS. **1** and **2** for exemplary purposes, it should be recognized that the method described with reference to FIG. **3** is certainly not limited to the specific embodiments described with reference to FIGS. **1** and **2**. As shown, in this embodiment a scan of a computer (e.g., the computer **100**) for pestware is initially carried out (Block **302**). In several embodiments for example, the detection engine **102** scans the file storage device **118**, the operating system registry and executable memory of the protected computer for indicia of pestware (e.g., processes or files matching pestware definitions and/or alterations to a registry entry that are consistent with pestware).

As shown in FIG. **3**, if pestware indicia is found, then recorded information (e.g., the activity log **120** and/or the N files **130**) that may include traces of the pestware is accessed (Block **304**), and traversed to search for information leading to the identification of the source of the pestware (Block **306**). In some instances, the origin of the pestware may be identified by simply referencing one piece of data that is stored in connection with the identification of the pestware source. In other instances, however, it may be necessary to access several pieces of referential data, which may be located in one or more of the files **220-256** before arriving at the identity of the source of the pestware.

As an example, if a pestware file is found on the file storage device **118**, then the cache index **243** of the browser cache **242** may be searched to identify the name of the file, and if the originating URL is stored in connection with the file, the URL is identified as the source of the file. As another example, if a pestware process is identified, a search of the activity log **220** may lead to the identity of a second process that spawned the pestware process, and additional searching of the activity log **220** using the name of the second process may lead to the identification of a pestware file associated with the second process. In turn, a search of the cache index **243** for the name of the pestware file may lead to the URL from which the pestware file was downloaded.

As shown in FIG. **3**, if the source of the pestware is identified (Block **308**), then the source of the pestware is reported (Block **310**). In some embodiments, it is contemplate that several other computers configured in accordance with the protected computer **100** depicted in FIG. **1** may also report sources of pestware to a centralized location where the URLs may be added to a list of "bad URLs." In addition, the identified URLs may be actively searched to learn more about the pestware generated at the sites, which may help generate definitions for the pestware and may provide information about how pestware infections occur.

Referring next to FIG. **4**, shown is a flowchart depicting a method in accordance with another embodiment of the present invention. Again, reference will be made to FIGS. **1** and **2** for exemplary purposes, but it should be recognized that the method described with reference to FIG. **4** is certainly not limited to the specific embodiments described with reference to FIGS. **1** and **2**. As shown, in this embodiment activity on a computer is monitored for indicia of pestware (Block **402**), and if potential pestware-related activity is detected, recorded information related to the activity is searched to identify one or more suspicious objects (e.g., files and/or processes) that are related to the activity (Block **404**). In some embodiments, the shields **120** (described with reference to FIG. **1**) may monitor the protected computer **100** for activities, and if the activity is identified as potential pestware activity (e.g., by the heuristics module **110**), then the research module **108** searches the activity log **120** and/or one or more of the N files for information relating to the pestware-related activity.

As shown in FIG. **4**, recorded information (e.g., one or more of the N files) is then traversed to trace through information related to the suspicious objects that leads to an origin of the suspicious objects (Block **406**). In one embodiment for example, the suspicious activity leads to a search for suspicious processes and/or files related to the activity (e.g., using the activity log **120**), which then leads to a search of one or more of the N files **130** (e.g., the cache index **243**) for an indication of the source of the suspicious process and/or files.

As depicted in FIG. **4**, if the source of the suspicious object(s) is identified (Block **408**), then the source of the suspicious object(s) is then reported (e.g., to a pestware research entity). In this way, the suspicious objects and the web sites originating the suspicious objects may be further researched to establish the extent to which they may be a threat.

As an example of pestware-related activity that may trigger the search for a source of the activity, if a series of particular API calls is made in a pattern known to be associated with pestware, the process(es) making the calls may be identified using, for example, the activity log **120**. In turn, the activity log **120** may be used to identify the file(s) associated with the process(es), and the cache index **243** may be utilized to search for a source of the file(es). It should be recognized that this is merely one example of the type of activity that may trigger backwards researching of logs on a computer, and that patterns in process creation, downloaded files, changes to an operating system registry and browser settings, for example, may trigger a search of the computer's logs.

Referring next to FIG. **5**, shown is a flowchart depicting yet another method in accordance with another embodiment of the present invention. While referring to FIG. **5**, simultaneous reference will be made to FIGS. **1** and **2** for exemplary purposes, but it should be recognized that the method described

with reference to FIG. **5** is certainly not limited to the specific embodiments described with reference to FIGS. **1** and **2**.

As shown, in this embodiment a time of interest is initially established based upon a suspicion that pestware has infected a computer (Block **502**). In some embodiments, for example, a user may establish the time of interest based upon events the user observed (e.g., pop-ups or a system crash). In one embodiment, as discussed with reference to FIG. **1**, in response to a user request, the time stamp module **112** may issue a time stamp and initiate research that is related to activity occurring at or around the time of the time stamp. In variations, the user is provide with an alert in response to the heuristics module **110** identifying an activity that is suspicious and the user is given an option to initiate research at or around the time of interest.

In other embodiments, the time stamp module **112** automatically generates a time stamp in response to a report of suspicious activity (e.g., from the shields **120** or heuristics module **110**).

As shown in FIG. **5**, once a time of interest is established, suspicious activity is identified on the computer based upon the time of interest (Block **504**). The time of interest may be established, for example, to include a time period before the time stamp is issued so that a search for suspicious activity is limited to a particular time period. In some embodiments, the activity log **130** and/or one or more of the N files **130** are accessed and analyzed to determine whether any activity during the time of interest is suspicious (e.g., the activity indicates in some way that it may be associated with pestware). As an example, if any logged information (e.g., in the activity log **130** and/or one or more of the N files **130**) indicates that during the time of interest that, for example, access to the registry was carried out in connection with a downloaded file or the launch of a process, the activities may identified as being suspect and further research relative to the process and the file may carried out.

Beneficially, many of the logs accessed include time-stamped information, which enables an activity that occurred during the time of interest to be readily identified and analyzed either alone or in connection with other activities occurring during the time of interest. As an example, the activity log **220**, the browser history **240**, browser cache **242**, operating system setting **246**, the event log **248**, the debugging log **250** the firewall log **252**, the file information **254** and the monitoring software logs **256** include time stamped information that provides insight into the activities that occurred on the computer during the time of interest.

As depicted in FIG. **5**, once one or more activities are identified as being suspicious (Block **504**), one or more objects (e.g., processes or files) on the computer are associated with the suspicious activity (Block **506**). For example, the research module **108** may search the activity log **120** and/or one or more of the N files for information that associates the suspicious activity to one or more processes and the processes may be related to one or more files.

As shown in FIG. **5**, recorded information on the computer is then traversed in order to trace to an origin of one or more of the objects (Block **508**). For example, a search of one or more of the N files **130** (e.g., the cache index **243**) may be carried out to identify the source of a suspicious process and/or files. Once the source of the suspicious object(s) is identified (Block **510**), the source is then reported (e.g., to a remote research entity)(Block **512**).

It should be recognized that the methods described with reference to FIGS. **3**, **4** and **5** are merely exemplary and are certainly not the only modes of operation that are contemplated. As an example, the establishment of a time of interest, as discussed with reference to FIG. **5**, may be useful in the method described with reference to FIGS. **3** and **4** for identifying information that leads to the source of the pestware or pestware-related activities. Moreover, it is contemplated that aspects from all three of the methods described with reference to FIGS. **3**, **4** and **5** may be combined.

In conclusion, the present invention provides, among other things, a system and method for identifying a source of pestware or suspected pestware on a computer. Those skilled in the art can readily recognize that numerous variations and substitutions may be made in the invention, its use and its configuration to achieve substantially the same results as achieved by the embodiments described herein. Accordingly, there is no intention to limit the invention to the disclosed exemplary forms. Many variations, modifications and alternative constructions fall within the scope and spirit of the disclosed invention as expressed in the claims.

What is claimed is:

1. A method for identifying an origin of activity on a computer that is indicative of pestware comprising:

monitoring, using a kernel-mode driver, API call activity on the computer;

storing information related to the API call activity in a log;

analyzing, heuristically, the API call activity to determine whether one or more weighted factors associated with the API call activity exceeds a threshold;

identifying, based upon the API call activity, a suspected pestware object on the computer;

identifying, in response to the identifying the suspected pestware object, a reference to an identity of an externally networked source of the suspected pestware object; and

reporting the identity of the externally networked source to an externally networked pestware research entity, wherein

the identity of the externally networked source is selected from a group consisting of an I.P. address, a URL, an email client and a program.

2. The method of claim **1**, wherein the identifying the suspected pestware object includes accessing an activity log that includes information that relates the activity to the suspected pestware object.

3. The method of claim **1**, wherein the identifying a reference includes accessing at least a portion of a recorded history of externally networked sources.

4. The method of claim **3**, wherein the recorded history resides in at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

5. A system for identifying a source of activity on a computer that is indicative of pestware including:

a kernel-level monitor configured to monitor API call activity on the computer and to store information related to the API call activity in a log;

a heuristics module configured to analyze the API call activity to determine whether one or more weighted factors associated with the API call activity exceeds a threshold;

a research portion configured to identify, in response to a prompt from the heuristics module, a suspected pestware object on the computer; and

a reporting portion configured to generate a report including a reference to an identity of an externally networked source of the suspected pestware object and to report the

identity of the externally networked source to an externally networked pestware research entity, wherein

the kernel-level monitor is configured to store information about a relationship between at least one API call of the API call activity and a file.

6. The system of claim **5**, wherein the information comprises an identity of at least one process that made at least one API call in the log.

7. The system of claim **5**, wherein the heuristics module is configured to receive information about the API call activity from the kernel-level monitor and to determine whether the API call activity is indicative of pestware.

8. The system of claim **5**, wherein the kernel-level monitor is configured to intercept at least one API call.

9. The system of claim **5**, wherein the identity of the externally networked source is selected from the group consisting of an I.P. address, a URL, an email client and a program name.

10. The system of claim **5**, wherein the research portion is configured to access at least a portion of a recorded history of externally networked sources.

11. The system of claim **10**, wherein the recorded history resides in at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

12. A non-transitory computer-readable medium including processor-executable instructions for identifying an origin of activity on a computer that is indicative of pestware, the instructions including instructions for:

monitoring, using a kernel-mode driver, API call activity on the computer;

storing information related to the API call activity in a log;

analyzing, heuristically, the API call activity to determine whether one or more weighted factors associated with the API call activity exceeds a threshold;

identifying, based upon the API call activity, a suspected pestware object on the computer;

identifying, in response to the identifying the suspected pestware object, a reference to an identity of an externally networked source of the suspected pestware object; and

reporting the identity of the externally networked source to an externally networked pestware research entity, wherein

the identity of the externally networked source is selected by an identifier selected from a group consisting of an I.P. address, a URL, an email client and a program.

13. The non-transitory computer-readable medium of claim **12**, wherein the instructions for identifying the suspected pestware object include instructions for accessing an activity log that includes information that relates the activity to the suspected pestware object.

14. The non-transitory computer-readable medium of claim **12**, wherein the instructions for identifying a reference include instructions for accessing at least a portion of a recorded history of externally networked sources.

15. The non-transitory computer-readable medium of claim **14**, wherein the recorded history resides in at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

* * * * *

Exhibit 13

U 8164950

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

October 07, 2021

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THIS OFFICE OF:

U.S. PATENT: *8,763,123*

ISSUE DATE: *June 24, 2014*

By Authority of the

Under Secretary of Commerce for Intellectual Property and Director of the United States Patent and Trademark Office

R GLOVER

Certifying Officer

US008763123B2

(12) **United States Patent**
Morris et al.

(10) **Patent No.: US 8,763,123 B2**
(45) **Date of Patent: Jun. 24, 2014**

(54) **METHODS AND APPARATUS FOR DEALING WITH MALWARE**

(75) Inventors: **Melvyn Morris**, Turnditch (GB); **Paul Stubbs**, Wyboston (GB); **Markus Hartwig**, Milton Keynes (GB); **Darren Harter**, Hucclecote (GB)

(73) Assignee: **Prevx Limited** (GB)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **13/543,865**

(22) Filed: **Jul. 8, 2012**

(65) **Prior Publication Data**

US 2012/0278891 A1 Nov. 1, 2012

Related U.S. Application Data

(62) Division of application No. 11/477,807, filed on Jun. 30, 2006, now Pat. No. 8,418,250.

(30) **Foreign Application Priority Data**

Jun. 30, 2005 (GB) 0513375.6

(51) **Int. Cl.**
G06F 11/00 (2006.01)

(52) **U.S. Cl.**
USPC **726/23**; 726/1; 726/24; 726/25; 713/188; 700/200

(58) **Field of Classification Search**
USPC 726/1, 23, 24
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,338,141 B1 1/2002 Wells
6,772,346 B1 8/2004 Chess
6,944,772 B2 9/2005 Dozortsev
7,013,483 B2 3/2006 Cohen et al.
7,093,239 B1 8/2006 van der Made
7,370,359 B2 * 5/2008 Hrabik et al. 726/23
7,603,374 B2 10/2009 Cameron et al.
7,793,338 B1 * 9/2010 Beddoe et al. 726/3
7,979,889 B2 * 7/2011 Gladstone et al. 726/1
8,046,835 B2 * 10/2011 Herz 726/25
8,407,795 B2 * 3/2013 Palagummi 726/24
2001/0052014 A1 12/2001 Sheymov et al.

(Continued)

FOREIGN PATENT DOCUMENTS

EP 1315066 5/2003
EP 1280040 3/2004

(Continued)

OTHER PUBLICATIONS

Vasudevan A, Cobra —fine grained malware analysis, May 2006, IEEE, vol. 4, pp. 279-294.*

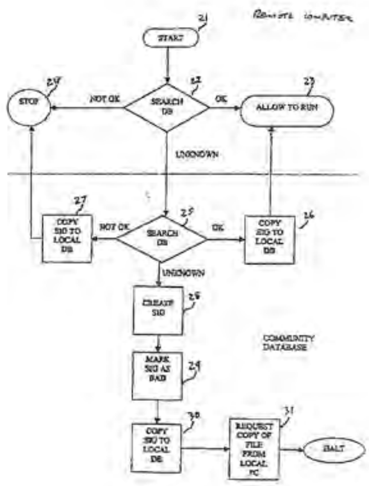
(Continued)

Primary Examiner — Cordelia Zecher
Assistant Examiner — Viral Lakhia
(74) *Attorney, Agent, or Firm* — Sheridan Ross P.C.

(57) **ABSTRACT**

In one aspect, a method of determining the protection that a remote computer has from malware includes receiving at a base computer, details of all or selected security products operating on a remote computer, receiving similar information from other remote computers, and identifying malware process that were not identified by the security products installed on the other remote computers and having a same or similar combination of security products installed on the remote computer.

16 Claims, 3 Drawing Sheets



Copy provided by USPTO from the PIRS Image Database on 10-05-2021

(56) **References Cited**

U.S. PATENT DOCUMENTS

2002/0087734	A1	7/2002	Marshall et al.	
2002/0099952	A1	7/2002	Lambert et al.	
2002/0194490	A1	12/2002	Halperin et al.	
2003/0023857	A1	1/2003	Hinchlife	
2003/0084323	A1	5/2003	Gales	
2003/0101381	A1 *	5/2003	Mateev et al.	714/38
2003/0131256	A1	7/2003	Ackroyd	
2003/0135791	A1	7/2003	Natvig	
2003/0195861	A1 *	10/2003	McClure et al.	707/1
2004/0006704	A1 *	1/2004	Dahlstrom et al.	713/200
2004/0039921	A1	2/2004	Chuang	
2004/0068618	A1	4/2004	Hooker	
2004/0073810	A1	4/2004	Dettinger et al.	
2004/0083129	A1 *	4/2004	Herz	705/10
2004/0083408	A1	4/2004	Spiegel	
2004/0153644	A1	8/2004	McCorkendale	
2004/0255165	A1	12/2004	Szor	
2005/0021994	A1	1/2005	Barton	
2005/0022014	A1	1/2005	Shipman	
2005/0027686	A1	2/2005	Shipp	
2005/0086500	A1	4/2005	Albornoz	
2005/0210035	A1	9/2005	Kester et al.	
2006/0085857	A1	4/2006	Omote et al.	
2007/0016953	A1	1/2007	Morris et al.	
2008/0209562	A1	8/2008	Szor	
2008/0320595	A1	12/2008	van der Made	
2009/0271867	A1	10/2009	Zhang	
2012/0278895	A1	11/2012	Morris et al.	
2013/0042294	A1 *	2/2013	Colvin et al.	726/1

FOREIGN PATENT DOCUMENTS

EP	1536341	6/2005
EP	1549012	6/2005
JP	3-233629	10/1991
JP	6-110718	4/1994
JP	9-504395	4/1997
JP	2003-196112	7/2003
WO	WO 95/12162	5/1995
WO	WO 96/30829	10/1996
WO	WO 99/15966	4/1999
WO	WO 02/33525	4/2002
WO	WO 03/021402	3/2003
WO	WO 2004/097602	11/2004

OTHER PUBLICATIONS

U.S. Appl. No. 60/789,156, filed Apr. 5, 2006, Chiriac.
English Language Abstract of JP 3-233629 published Oct. 17, 1991.
English Language Abstract and Translation of JP 6-110718 published Apr. 22, 1994.
English Language Abstract of JP 2003-196112 published Jul. 11, 2003.
U.S. Appl. No. 11/694,261.
Zenkin "Fighting Against the invisible Enemy—Methods for detecting an unknown virus," Computers & Security, Elsevier Science Publishers, Amsterdam, NL, vol. 20, No. 4, Jul. 31, 2001, pp. 316-321, XP004254268.
International Search Report issued in PCT/GB2006/002439 issued Jan. 15, 2007.
English Translation of Official Action for China Patent Application No. 201110036121.8, dated Sep. 24, 2012, 8 pages.
Official Action with English translation for China Patent Application No. 201110036121.8, dated Oct. 10, 2011, 14 pages.
Japanese Office Action issued in JP 2008-518975 on Sep. 13, 2011.
English Language Translation of Japanese Office Action issued in JP 2008-518975 on Sep. 13, 2011.
Official Action for U.S. Appl. No. 11/477,807, mailed Jan. 13, 2010, 8 pages.
Official Action for U.S. Appl. No. 11/477,807, mailed May 4, 2010, 13 pages.
Official Action for U.S. Appl. No. 11/477,807, mailed Sep. 28, 2010, 11 pages.
Official Action for U.S. Appl. No. 11/477,807, mailed Feb. 15, 2011, 13 pages.
Official Action for U.S. Appl. No. 11/477,807, mailed Jul. 25, 2011, 13 pages.
Notice of Allowance for U.S. Appl. No. 11/477,807, mailed Mar. 21, 2012, 9 pages.
Notice of Allowance for U.S. Appl. No. 11/477,807, mailed Dec. 13, 2012, 7 pages.
Official Action for China Patent Application No. 201110036124.1, dated Apr. 17, 2013, 7 pages.
Official Action for U.S. Appl. No. 13/543,866, mailed Mar. 22, 2013, 14 pages.
Official Action with English Translation for China Patent Application No. 20110036124.1, dated Oct. 23, 2013, 8 pages.
Notice of Allowance for U.S. Appl. No. 13/543,866, mailed Dec. 26, 2013 10 pages.
Official Action for European Patent Application No. 06755686.0 dated Jul. 31, 2013, 5 pages.
Extended European Search Report for European Patent Application No. 13167434.3 dated Aug. 1, 2013, 6 pages.
Extended European Search Report for European Patent Application No. 13167436.8 dated Aug. 30, 2013, 5 pages.
Official Action for U.S. Appl. No. 13/543,866, mailed Jul. 12, 2013, 12 pages.

* cited by examiner

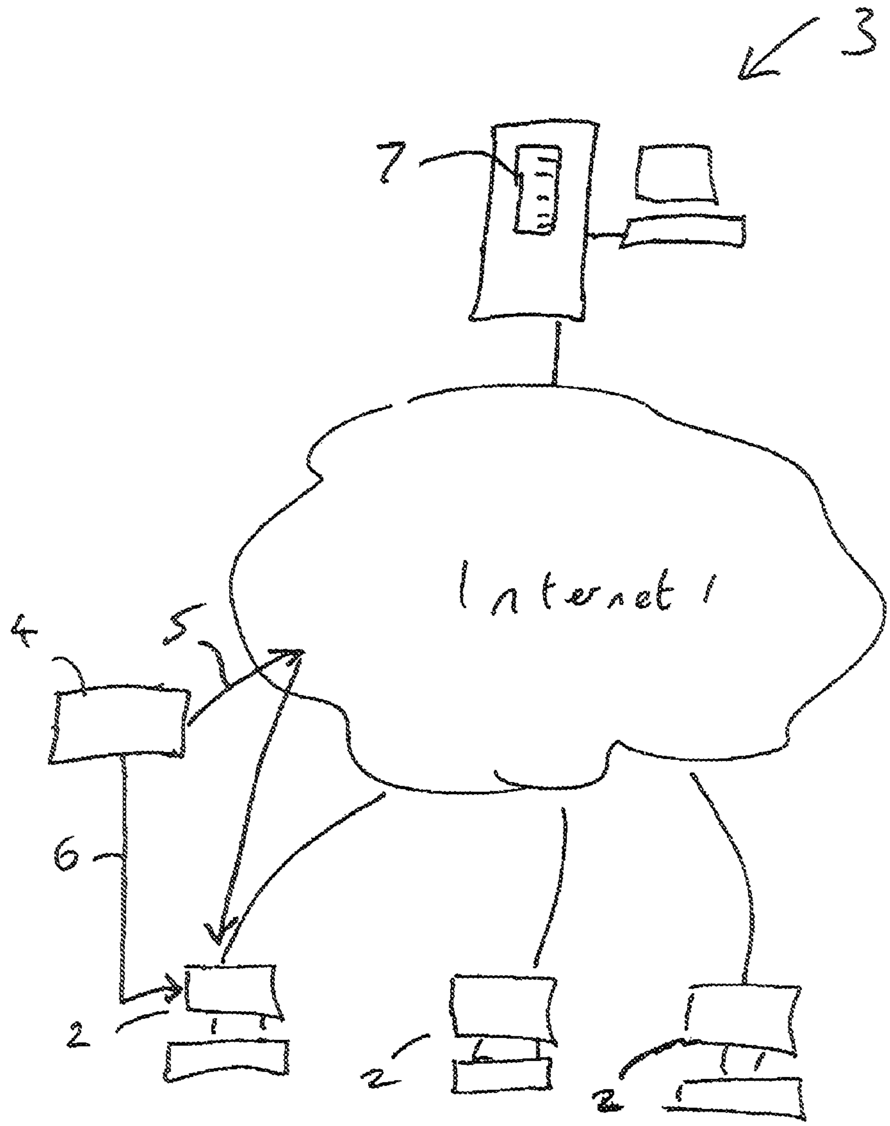


FIG. 1

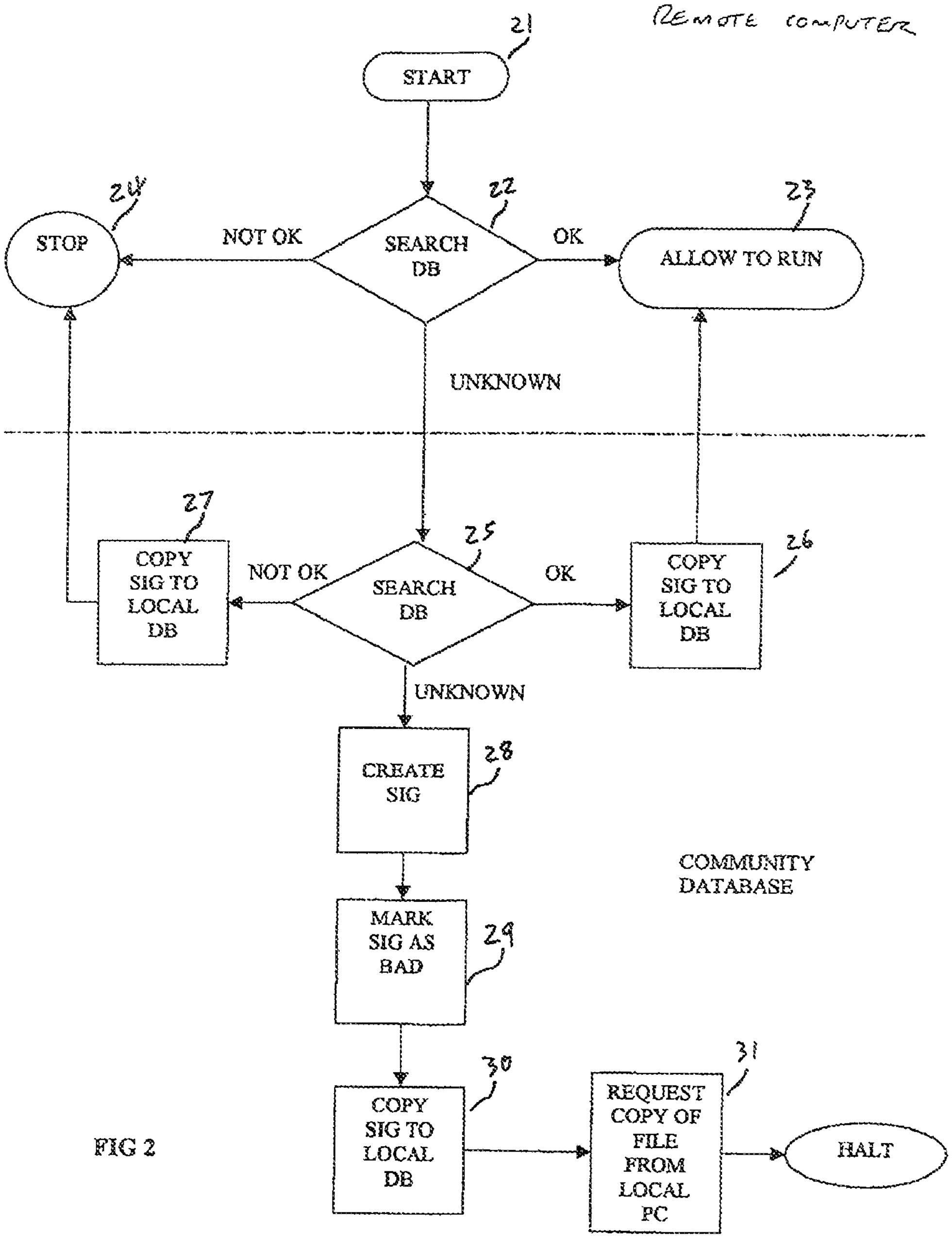

REMOTE COMPUTER
21
START
24
STOP
NOT OK
22
SEARCH DB
OK
23
ALLOW TO RUN
UNKNOWN
27
COPY SIG TO LOCAL DB
NOT OK
25
SEARCH DB
OK
26
COPY SIG TO LOCAL DB
UNKNOWN
CREATE SIG
28
COMMUNITY DATABASE
MARK SIG AS BAD
29
30
31
COPY SIG TO LOCAL DB
REQUEST COPY OF FILE FROM LOCAL PC
HALT

FIG 2

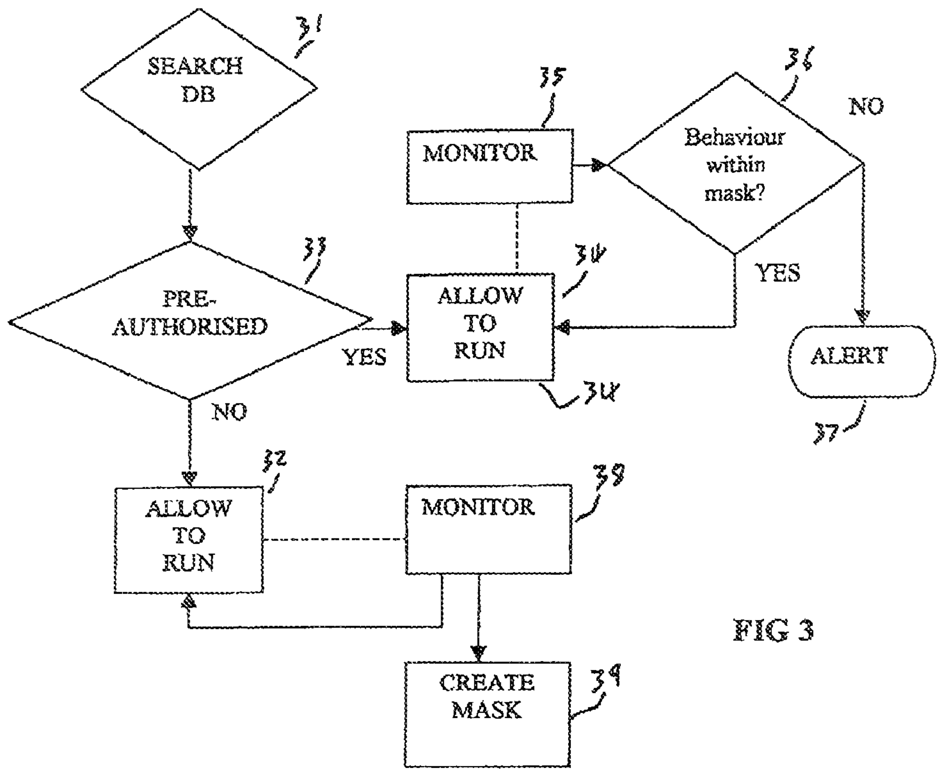

SEARCH DB
31
35
MONITOR
36
Behaviour within mask?
NO
YES
33
PRE-AUTHORISED
YES
34
ALLOW TO RUN
34
ALERT
37
NO
32
ALLOW TO RUN
38
MONITOR
CREATE MASK
39

FIG 3

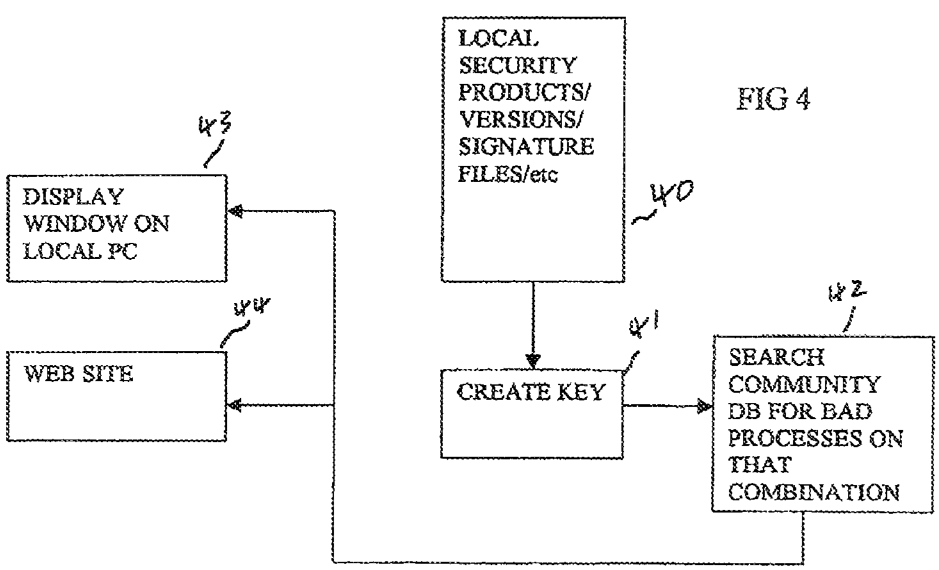

LOCAL SECURITY PRODUCTS/ VERSIONS/ SIGNATURE FILES/etc
40
43
DISPLAY WINDOW ON LOCAL PC
44
WEB SITE
41
CREATE KEY
42
SEARCH COMMUNITY DB FOR BAD PROCESSES ON THAT COMBINATION

FIG 4

METHODS AND APPARATUS FOR DEALING WITH MALWARE

This application is a divisional of U.S. application Ser. No. 11/477,807 filed Jun. 30, 2006, which claims priority to United Kingdom Application No. 0513375.6, filed Jun. 30, 2005. The entirety of all of the above-listed Applications are incorporated herein by reference.

The present invention relates generally to methods and apparatus for dealing with malware. In one aspect, the present invention relates to a method and apparatus for classifying a computer object as malware. In another aspect, the present invention relates to a method and apparatus for determining the protection that a remote computer has from malware. In another aspect, the present invention relates to a method and apparatus for classifying a computer object as malware or as safe. In another aspect, the present invention relates to a method of installing software on a computer.

The term "malware" is used herein to refer generally to any executable computer file or, more generally "object", that is or contains malicious code, and thus includes viruses, Trojans, worms, spyware, adware, etc. and the like.

A typical anti-malware product, such as anti-virus scanning software, scans objects or the results of an algorithm applied to the object or part thereof to look for signatures in the object that are known to be indicative of the presence of a virus. Generally, the method of dealing with malware is that when new types of malware are released, for example via the Internet, these are eventually detected. Once new items of malware have been detected, then the service providers in the field generate signatures that attempt to deal with these and these signatures are then released as updates to their anti-malware programs. Heuristic methods have also been employed.

These systems work well for protecting against known malicious objects. However, since they rely on signature files being generated and/or updated, there is inevitably a delay between a new piece of malware coming into existence or being released and the signatures for combating that malware being generated or updated and supplied to users. Thus, users are at risk from new malware for a certain period of time which might be up to a week or even more. Moreover, in order to try to defeat anti-virus products, the malware writers use obfuscation techniques in order to attempt to hide the signature or signature base data of the virus code from detection. Typically, the obfuscation involves encrypting or packing the viral code.

WO-A-2004/097602 describes a system that analyses computer files received or generated by a local computer and compares these with a database of known files to determine whether a particular file is known and if so whether it has been known about long enough that it can be regarded as "safe". However, in practice, on its own this is not likely to provide for adequate protection because, for example, the active payload of a virus or Trojan may only be programmed to activate at a particular date, or upon receiving a message or instruction from a local or remote system or process, or on the occurrence of a particular event that may be many months or even years after the process has been first run or is released. Thus, just looking at the age of a file is an unsatisfactory way of determining whether it is properly safe and will remain so.

In the system of US-A-2004/0083408, a worm in a file is detected by examining connection attempts made by the specific file running on a computer.

U.S. Pat. No. 6,944,772, U.S. Pat. No. 6,772,346, EP-A-1549012 and EP-A-1280040 all disclose "community-based" anti-malware systems in which a plurality of "local"

computers all connect via a network (which may be a LAN or the Internet, for example) to a central computer. On encountering a file that is not already known to them, the local computers send a request to the central computer for authorisation to run the file. If the file is recognised at the central computer, then the central computer can send permission for the local computer to run the file if the file is known to be safe or send a "deny" command if the file is known to be malicious. However, in each of these prior art proposals, if the file is not known at the central computer, then the whole file is sent to the central computer where it can be analysed to determine whether it should be regarded as safe or malware. Such analysis is typically carried out manually or "semi-manually" by subjecting the file to detailed analysis, for example by emulation or interpretation, which can still take days given the human involvement that is typically required. There is therefore still a considerable period of time before a new file is classified as safe or as malware. In the case of these prior art systems, the request for authorisation to run the file that is sent by a local computer to the central computer may comprise sending a checksum or "signature" or "key" that uniquely represents the file.

A similar community-based anti-malware system is disclosed in WO-A-02/33525. In this system, in the case that a local computer is seeking clearance to run a file that is not known by the central computer to be safe or malware, some limited audit information about the prevalence of the file on other local computers can be sent to a human system administrator associated with the local computer that is seeking permission to run the file. The human system administrator can therefore make a better informed though still "manual" decision as to whether or not the file is safe to run.

In the system of US-A-2004/0073810, a metafile containing data about an attachment or other transmitted file is sent to a central computer. The data about that file is analysed to determine a likelihood of the transmitted file being malware. A specific example given is that if the transmitted file has been transmitted via at least a certain number of servers, then it should be treated as malware.

In the systems disclosed in US-A-2005/0021994 and US-A-2004/0153644, pre-approved files, which may be certified as safe by for example the software vendor associated with the files, may be permitted always to run without further checking. In one embodiment of the system of US-A-2004/0153644, monitoring is carried out to decide that a file is malicious if an abnormally high number of requests by that file is received at a central authority from plural local computers in a time period or if an abnormally high number of requests by that file on a single computer is received from the single local computer in a time period.

In the system of US-A-2004/0006704, a comparison is made between installed versions of software on a computer with a database of software versions and their known vulnerabilities. A user of the computer can therefore be informed of specific risks and how to minimise those risks by updating existing or installing new software.

In the system of WO-A-03/021402, a central database holds a virtual image of all files stored on each of plural local computers. If a threat in one local computer is identified, other local computers with a similar configuration can be notified of the risk.

Thus, the prior art systems either rely on deep analysis of a new object in order to determine whether or not the object is malicious, which introduces delay and therefore risk to users during the period that the file is analysed and new anti-malware, signatures distributed, or limited analysis of the opera-

tion of the particular object or its method of transmission to a computer is carried out to decide a likelihood of the object being malicious.

According to a first aspect of the present invention, there is provided a method of classifying a computer object as malware, the method comprising:

at a base computer, receiving data about a computer object from each of plural remote computers on which the object or similar objects are stored;

comparing in the base computer the data about the computer object received from the plural computers; and,

classifying the computer object as malware on the basis of said comparison.

Compared to the prior art that relies solely on signature matching, this aspect allows a comparison to be made between the objects and/or their effects on the different remote computers to determine whether or not a particular object should be classed as good or as malware. Sophisticated pattern analysis can be carried out. This allows a rapid determination of the nature of the object to be made, without requiring detailed analysis of the object itself as such to determine whether it malware and also avoids the need to generate new signatures to be used for signature matching as in the conventional prior art anti-virus software.

In a preferred embodiment, the data about the computer object that is sent from the plural remote computers to the base computer and that is used in the comparison includes one or more of: executable instructions contained within or constituted by the object; the size of the object; the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

Preferably, the comparing identifies relationships between the object and other objects. In an example, this can be used immediately and automatically to mark a child object as bad (or good) if the or a parent or other related object is bad (or good). Thus, if at least one other object to which said object is related is classed as malware, then the method may comprise classifying said object as malware. Said other objects include the object or similar objects stored on at least some of the remote computers. Said other objects may include other objects that are parent objects or child objects or otherwise process-related objects to said object.

In a most preferred embodiment, the data is sent in the form of key that is obtained by a hashing process carried out in respect of the objects on the respective remote computers. A major advantage of using such a key is that it keeps down the volume of data that needs to be transmitted to the base computer. Given that there may be thousands or even millions of connected remote computers and further given that each may send details about very many objects, this can be an important advantage.

The key preferably has at least one component that represents executable instructions contained within or constituted by the object. This important preferred feature allows a comparison to be made at the base computer of only the executable instructions of the object. This means for example that differently named objects that basically have the same executable instructions, which is often an indicator that the objects are malware, can nevertheless be regarded as the "same" object for this purpose. As another example, a new version of a program may be released which has minor changes compared to a previous version already known to the base computer and which in substance, at least in respect of the executable instructions, can be regarded as being the same as the previous version. In that case, the minor differences can be ignored and the objects regarded as being the same. Not only is this useful in distinguishing between malware and for example revised versions of previous software, it also keeps down the data transmission and storage requirements because the base computer can inform the remote computers that an apparently new object is for this purpose the same as a previously known object, thus avoiding having the remote computers send full details about the object or the object itself to the base computer.

The key preferably has at least one component that represents data about said object. Said data about said object may include at least one of: the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

The key preferably has at least one component that represents the physical size of the object.

Where more than one of these components are present in the key, the plural components are preferably severable.

The method may comprise initially classifying an object as not malware, generating a mask for said object that defines acceptable behaviour for the object, and comprising monitoring operation of the object on at least one of the remote computers and reclassifying the object as malware if the actual monitored behaviour extends beyond that permitted by the mask. This provides an efficient and effective way of monitoring the behaviour of an object that has been classified or regarded as good and allows the object to be reclassified quickly as malware if the behaviour of the object warrants it.

According to a second aspect of the present invention, there is provided apparatus for classifying a computer object as malware, the apparatus comprising:

a base computer constructed and arranged to receive data about a computer object from each of plural remote computers on which the object or similar objects are stored;

the base computer being constructed and arranged to compare the data about the computer object received from said plural computers; and,

the base computer being constructed and arranged to classify the computer object as malware on the basis of said comparison.

According to a third aspect of the present invention, there is provided a method of providing data about a computer object from a remote computer to a base computer so that a comparison can be made at the base computer with similar data received from other remote computers, the method comprising:

providing from a remote computer to a base computer data about a computer object that is stored on the remote computer;

the data including one or more of: executable instructions contained within or constituted by the object; the size of the object; the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers;

the data being sent in the form of key that is obtained by a hashing process carried out in respect of the object on the remote computer.

This method, which may be carried out by so-called agent software running on the remote computer, allows for efficient sending of data to the base computer, which minimises data transmission and storage requirements and also permits rapid analysis to be made at the base computer.

The key preferably has at least one component that represents executable instructions contained within or constituted by the object.

The key preferably has at least one component that represents data about said object. Said data about said object may include at least one of: the current name of the object; the physical and folder location of the object on disk; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer;

and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers.

The key preferably has at least one component that represents the physical size of the object.

According to a fourth aspect of the present invention, there is provided a method of determining the protection that a remote computer has from malware, the method comprising:

receiving at a base computer details of all or selected security products operating at a point in time on said remote computer;

receiving similar information from other remote computers connected to the base computer; and,

identifying any malware processes that were not identified by said other remote computers having that particular combination of security products.

In this way, the base computer can be used to obtain information as to whether for example a particular, specific combination of operating system and various security products, including settings and signature files existing at a point in time, renders a particular computer having those products and settings susceptible or vulnerable to any particular malware object. The user can be advised accordingly and for example provided with recommendations for remedying the situation.

The method may therefore comprise providing information to the user of said remote computer that said remote computer may be susceptible to attack by said malware processes on the basis of said identifying.

The details of all or selected security products preferably includes the name of the security products, versions, and loaded signature files.

According to a fifth aspect of the present invention, there is provided apparatus for determining the protection that a remote computer has from malware, the apparatus comprising:

a base computer constructed and arranged to receive computer details of all or selected security products operating at a point in time on said remote computer;

the base computer being constructed and arranged to receive similar information from other remote computers connected to the base computer; and,

the base computer being constructed and arranged to identify any malware processes that were not identified by said other remote computers having that particular combination of security products.

According to a sixth aspect of the present invention, there is provided a method of classifying a computer object as malware or as safe, wherein said computer object is a descendant or otherwise related object of a first computer object, the method comprising:

classifying a first computer object as malware or as safe;

identifying in a key relating to said first computer object a component that uniquely identifies the first computer object and that is inherited or otherwise present in the key of a descendant or other related computer object of the first computer object; and,

classifying said computer object as malware or as safe as the case may be on the basis of the unique identifier component being present in the key of said computer object.

This aspect uses the concept of ancestry to enable objects to be marked as malware. For example, any particular process may spawn child processes which are therefore related. The key relating to the first object may be inspected to identify a component that uniquely identifies the first object and that is inherited or otherwise present in the key of a descendant or other related object of the first object.

The method may comprise monitoring activities of said first computer object and reclassifying the first computer object as malware in the case that it was initially classified as safe and subsequently determined to be malware, the method further comprising automatically classifying as malware any computer object that has a key in which said unique identifier component is present.

According to a seventh aspect of the present invention, there is provided apparatus for classifying a computer object as malware or as safe, wherein said computer object is a descendant or otherwise related object of a first computer object, the apparatus comprising:

a computer constructed and arranged to classify a first computer object as malware or as safe;

the computer being constructed and arranged to identify in a key relating to said first computer object a component that uniquely identifies the first computer object and that is inherited or otherwise present in the key of a descendant or other related computer object of the first computer object; and,

the computer being constructed and arranged to classify said computer object as malware or as safe as the case may be on the basis of the unique identifier component being present in the key of said computer object.

According to an eighth aspect of the present invention, there is provided a method of installing software on a computer, the method comprising:

on initiation of installation of software on a computer, providing a computer-generated prompt on the computer to a user to ascertain whether the user authorises the installation; and,

ceasing the installation if a user authorisation is not received, else:

receiving at the computer the user's authorisation to proceed with the installation;

proceeding with the installation;

obtaining data about computer objects that are created or used during the installation;

storing said data at the local computer.

This provides for security when a user is installing new software and is not for example connected to a base computer having a community database of the type mentioned above. In that case, the method, which may be implemented in agent software running on the local computer, allows the user to permit the installation to proceed whilst at the same time gathering data about the objects (such as processes, new files, etc.) that are created during the installation.

Preferably, the locally stored data is referred to during the installation to ensure that all objects created or used during

the installation are part of the installation process, and, if it is found that objects created or used during the installation are not part of the installation process, either or both of: (i) ceasing the installation and (ii) providing a computer-generated prompt on the computer to the user accordingly. This allows the method to ensure that only those objects that are required for the installation are permitted to be created or used and thus avoids unwittingly allowing malware to install (given that malware often creates objects that are not expected as part of a normal installation of new software).

In a preferred embodiment, the method comprises connecting the computer to a community database that is connectable to a plurality of computers, and uploading the stored data to the community database for comparison with similar data provided by other computers:

The method may comprise downloading data about trusted installers to the computer, said data about trusted installers being referred to during the installation such that any objects relating to or created by the trusted installer are automatically authorised to proceed. This facilitates installation of software that is known a priori to be trustworthy.

Said data about trusted installers may be referred to only for a predetermined time period following receipt at the computer of the user's authorisation to proceed with the installation.

The present invention also includes computer programs comprising program instructions for causing a computer to perform any of the methods described above.

Although the embodiments of the invention described with reference to the drawings comprise computer processes performed in computer apparatus and computer apparatus itself, the invention also extends to computer programs, particularly computer programs on or in a carrier, adapted for putting the invention into practice. The program may be in the form of source code, object code, a code intermediate source and object code such as in partially compiled form, or in any other form suitable for use in the implementation of the processes according to the invention. The carrier be any entity or device capable of carrying the program. For example, the carrier may comprise a storage medium, such as a ROM, for example a CD ROM or a semiconductor ROM, or a magnetic recording medium, for example a floppy disk or hard disk. Further, the carrier may be a transmissible carrier such as an electrical or optical signal which may be conveyed via electrical or optical cable or by radio or other means.

Embodiments of the present invention will now be described by way of example with reference to the accompanying drawings, in which:

FIG. **1** shows schematically apparatus in which an embodiment of the present invention may be implemented;

FIG. **2** is a flowchart showing schematically the operation of an example of a method according to an embodiment of the present invention;

FIG. **3** is a flowchart showing schematically the operation of another example of a method according to an embodiment of the present invention; and,

FIG. **4** is a flowchart showing schematically an information obtaining stage.

Referring to FIG. **1**, a computer network is generally shown as being based around a distributed network such as the Internet **1**. The present invention may however be implemented across or use other types of network, such as a LAN. Plural local or "remote" computers **2** are connected via the Internet **1** to a "central" or "base" computer **3**. The computers **2** may each be variously a personal computer, a server of any type, a PDA, mobile phone, an interactive television, or any other device capable of loading and operating computer objects. An object in this sense may be a computer file, part of a file or a sub-program, macro, web page or any other piece of code to be operated by or on the computer, or any other event whether executed, emulated, simulated or interpreted. An object **4** is shown schematically in the figure and may for example be downloaded to a remote computer **2** via the Internet **1** as shown by lines **5** or applied directly as shown by line **6**.

In one preferred embodiment, the base computer **3** holds a database **7** with which the remote computers **2** can interact when the remote computers **2** run an object **4** to determine whether the object **4** is safe or unsafe. The community database **7** is populated, over time, with information relating to each object run on all of the connected remote computers **2**. As will be discussed further below, data representative of each object **4** preferably takes the form of a so-called signature or key relating to the object and its effects. As will also be discussed further below, the database **7** may further include a mask for the object **4** that sets out the parameters of the object's performance and operation.

Referring now to FIG. **2**, at the start point **21**, a computer object **4** such as a process is run at a remote computer **2**. At step **22**, by operation of local "agent" software running on the remote computer **2**, the operation of the process is hooked so that the agent software can search a local database stored at the remote computer **2** to search for a signature or key representing that particular process, its related objects and/or the event. If the local signature is present, it will indicate either that the process is considered to be safe or will indicate that that process is considered unsafe. An unsafe process might be one that has been found to be malware or to have unforeseen or known unsafe or malevolent results arising from its running. If the signature indicates that the process is safe, then that process or event is allowed by the local agent software on the remote computer **2** to run at step **23**. If the signature indicates that the process is not safe, then the process or event is stopped at step **24**.

It will be understood that there may be more than two states than "safe" or "not-safe" and choices may be given to the user. For example, if an object is considered locally to be not safe, the user may be presented with an option to allow the related process to run nevertheless. It is also possible for different states to be presented to each remote computer **2**. The state can be varied by the central system to take account of the location, status or ownership of the remote computer or timeframe.

If the object is unknown locally, then details of the object are passed over the Internet **1** or other network to the base computer **3** for storing in the community database **7** and preferably for further analysis at the base computer **3**. In that case, the community database **7** is then searched at step **25** for a signature for that object that has already been stored in the community database **7**. The community database **7** is supplied with signatures representative of objects, such as programs or processes, run by each monitored remote computer **2**. In a typical implementation in the field, there may be several thousands or even millions of remote computers **2** connected or connectable to the base computer **3** and so any objects that are newly released upon the Internet **1** or that otherwise are found on any of these remote computers **2** will soon be found and signatures created and sent to the base computer **3** by the respective remote computers **2**.

When the community database **7** is searched for the signature of the object that was not previously known at the remote computer **2** concerned, then if the signature is found and indicates that that object is safe, then a copy of the signature or at least a message that the object is safe is sent to the local

database of the remote computer **2** concerned at step **26** to populate the local database. In this way, the remote computer **2** has this information immediately to hand the next time the object **4** is encountered. A separate message is also passed back to the remote computer **2** to allow the object to run in the current instance.

If the signature is found in the community database **7** and this indicates for some reason that the object is unsafe, then again the signature is copied back to the local database and marked "unsafe" at step **27**, and/or a message is sent to the remote computer **2** so that running of the object is stopped (or it is not allowed to run) and/or the user given an informed choice whether to run it or not.

If after the entire community database **7** has been searched the object is still unknown, then it is assumed that this is an entirely new object which has never been seen before in the field. A signature is therefore created representative of the object at step **28**, or a signature sent by the remote computer **2** is used for this purpose, and this signature is initially marked as bad or unsafe community database **7** at step **29**. The signature is copied to the local database of the remote computer **2** that first ran the object at step **30**. A message may then be passed to the remote computer **2** to instruct the remote computer **2** not to run the object or alternatively the user may be given informed consent as to whether to allow the object to run or not. In addition, a copy of the object itself may be requested at step **31** by the community database **7** from the remote computer **2**.

If the user at the remote computer **2** chooses to run a process that is considered unsafe because it is too new, then that process may be monitored by the remote computer **2** and/or community database **7** and, if no ill effect occurs or is exhibited after a period of time of n days for example, it may then be considered to be safe. Alternatively, the community database **7** may keep a log of each instance of the process which is found by the many remote computers **2** forming part of the network and after a particular number of instances have been recorded, possibly with another particular number of instances or the process being allowed to run and running safely, the signature in the community database **7** may then be marked as safe rather than unsafe. Many other variations of monitoring safety may be done within this concept.

The details of an object **4** that are passed to the base computer **3** are preferably in the form of a signature or "key" that uniquely identifies the object **4**. This is mainly to keep down the data storage and transmission requirements. This key may be formed by a hashing function operating on the object at the remote computer **2**.

The key in the preferred embodiment is specially arranged to have at least three severable components, a first of said components representing executable instructions contained within or constituted by the object, a second of said components representing data about said object, and a third of said components representing the physical size of the object. The data about the object in the second component may be any or all of the other forms of identity such as the file's name, its physical and folder location on disk, its original file name, its creation and modification dates, vendor, product and version and any other information stored within the object, its file header or header held by the remote computer **2** about it; and, events initiated by or involving the object when the object is created, configured or runs on the respective remote computers. In general, the information provided in the key may include at least one of these elements or any two or more of these elements in any combination.

In one preferred embodiment, a check sum is created for all executable files, such as (but not limited to) .exe and .dll files, which are of the type PE (Portable Executable file as defined by Microsoft). Three types of checksums are generated depending on the nature of the file:

Type 1: five different sections of the file are check summed. These include the import table, a section at the beginning and a section at the end of the code section, and a section at the beginning and a section at the end of the entire file. This type applies to the vast majority of files that are analysed;

Type 2: for old DOS or 16 bit executable files, the entire file is check summed;

Type 3: for files over a certain predefined size, the file is sampled into chunks which are then check summed. For files less than a certain predefined size, the whole file is check summed.

For the check summing process, in principle any technique is possible. The MD5 (Message-Digest algorithm 5) is a widely-used cryptographic hash function that may be used for this purpose.

This allows a core checksum to be generated by viewing only the executable elements of the checksum and making a comparison between two executables that share common executable code.

For the type 1 checksum mentioned above, three signature processes may be used. The first defines the entire file and will change with almost any change to the file's content. The second attempts to define only the processing instructions of the process which changes much less. The third utilises the file's size, which massively reduces the potential of collisions for objects of differing sizes. By tracking the occurrences of all signatures individually appearing with different counterparts, it is possible to identify processes that have been changed or have been created from a common point but that have been edited to perform new, possibly malevolent functionality.

This "meta data" enables current and newly devised heuristics to be run on the data in the community database **7**.

The data stored in the community database **7** provides an extensive corollary of an object's creation, configuration, execution, behaviour, identities and relationships to other objects that either act upon it or are acted upon by it.

The preferred central heuristics use five distinct processes to establish if an object is safe, unsafe or suspicious.

The first of the said processes utilises the singularity or plurality of names, locations, vendor, product and version information captured and correlated from all of the remote computers **2** that have seen the object. By considering the plurality of this information for a single object, a score can be determined which can be used as a measure of the authenticity and/or credibility of the object. Most safe objects tend not to use a large plurality of identifying information or locations. Rules can be established to consider this information in respect of the type of object and its location. For example, temporary files often utilise a plurality of system generated file names which may differ on each remote computer for the same object. Where an object has little plurality, then it provides a reference point to consider its behaviour in comparison to the known behaviours of other objects that have previously used that identifying information. For example, a new object that purports to be a version of notepad.exe can have its behaviour compared with the behaviour of one or more other objects that are also known as notepad.exe. This comparison may be against a single other object or multiple other objects that use the same or even similar identifying information. In this way, new patterns of behaviour can be identified for the new object. Also it allows the preferred embodiment to police an object's behaviour over time to identify new behavioural patterns that may cause an object that was previously consid-

ered safe to have its status reconsidered. Alternatively, the score based on identities may be considered along with scores for other objects or scores created by other processes on this object to be considered in combinations.

The second of the said processes utilises an object's relationship to other objects that act upon it or upon which it acts. For example, analysis can be made of which object created this object, which objects this object created, which objects created a registry key to run or configure this object, which objects were configured by or had registry keys created by this object, etc. In this regard an object is considered to have a relationship based on the event performed by it upon another object, or upon it by another object. This simple 1-to-1 relationship chain provides a complex series of correlation points, allowing ancestral relationships to be considered for any object and its predecessors by event or its issue (i.e. child and sub-child processes) by event. This allows a score to be developed that describes its relationships and associations with known, unknown, known safe or known bad objects or a combination thereof. Objects that have specific relationships, volumes of relationships or mixes of relationships to one type or another may be judged safe or unsafe accordingly. Alternatively, the relationship-based score may be considered along with other scores to arrive at a determination of safe or unsafe. This data can also be used to deduce a number of factors about objects related directly or via other objects and their behaviours. For example it is possible to deduce how one object's behaviour can be influenced or changed by its association or linkage to another. Consider for example notepad.exe as supplied by Microsoft with the Windows series of operating systems. It has a limited range of functionality and would not be expected therefore to perform a wide variety of events, such as transmitting data to another computer or running other programs etc. However, the behaviour of notepad.exe could be modified by injecting new code into it, such as via dynamic link library injection (DLL injection). In this case notepad.exe would now have new capabilities derived by the code injection or linkage to another object. Using the data that defines the relationships between, objects it is possible to deduce that the new behaviours of a program can be attributed to the association with another object. If that new behaviour is malevolent, then it is possible to mark either or all processes as unsafe as appropriate.

The combination of behaviours captured provide a basis to determine if the object is safe or unsafe. Malware typically exhibit certain behaviour and characteristics. For example, malware frequently has a need to self-persist. This manifests itself in the need to automatically restart on system restarts or upon certain events. Creating objects in specific locations to auto restart or trigger execution is a typical characteristic of malware. Replacing core objects of the Windows system environment are another example of typical characteristics of malware. By providing a pattern of behaviour, the determination of objects to be unsafe or safe can be automated. The centralisation of the community data in the community database **7** provides the ability to rapidly assimilate object behaviours, allowing for the rapid identification and determination of malware. Objects may also perform events upon themselves which can be considered in deriving a score.

The third said process involves time and volumes. Relevant data includes when the object was first seen, when it was last seen, how many times it has been seen, how many times it has been seen in a given interval of time, and the increase or decrease of acceleration in it being seen. This information is highly relevant in determining the prevalence of an object in the community of remote computers. A score is developed based on these metrics which can be used to determine if an object is safe, unsafe or too prevalent to allow it to execute or propagate without very thorough examination. In this case, the object can be temporarily held or blocked from executing pending further information about its behaviour or relationships. This score may also be used in combination with scores from other processes. Time is also highly relevant in combination with other information, including but not limited to behaviours and identities. For example in the case of polymorphic or randomly named objects, time is a powerful qualifier. (A polymorphic virus changes its encryption algorithm and the corresponding encryption keys each time it replicates from one computer to another and so can be difficult to detect by conventional measures.) A program that creates other programs can often be considered normal or abnormal based on its activity over time.

The fourth said process considers the behaviour of an object. This allows a score to be developed based on the types of events performed by an object or events performed on it by itself or other objects. The centralised system of the community database **7** allows for an unlimited number of event types and can consider the object performing the event or the object having the event performed upon it, or both. Some event types also relate to external information other than objects, for example a program performing an event to connect with an Internet Chat Relay site, or a program modifying a non-executable file such as the Windows hosts file. The behavioural events of an object, be they as "actor" (i.e. the object doing something to another object) or as "victim" (i.e. the object has something done to it by another object) of any event can be considered in many ways, such as in combination, in sequence, in volume, in presence or absence, or in any combination thereof. The behavioural events in the preferred embodiment may have been provided by a remote computer **2** or from other external sources. The process can consider these in isolation or in combination. Furthermore it is a feature of the preferred embodiment that the behavioural events can be considered in combination with the status of other objects upon which the object acts or that act upon the object. For example, creating a program may have a different score if the program being created is safe, unsafe, new, unknown or suspicious. Similarly, a program that is created by a known bad program will likely have a different score attributed to its creation event depending on the status of the object creating it.

The fifth process considers the behaviour of a web page or script. In this respect, the web page and url combination is assigned a unique identity which allows its behaviour to be tracked as if it were an object like any other executable file. In this example, the web page may perform events that would normally be seen as events performed by the web browser (e.g. IExplore.exe or Firefox.exe). The preferred embodiment substitutes the identifying details and signatures of the web browser for the "pseudo" object identity associated with the web page being displayed or executing within the browser. In this respect, the status of the web page and/or web site to which it relates may be determined as safe, unsafe, unknown or suspicious in the same way as any other object. The web page's "pseudo" object identity also allows the preferred embodiment to block, interject or limit the functionality of that web page or web site to prevent some or all of its potentially unsafe behaviour or to provide the remote user with qualifying information to guide them about the safety of the web site, web page or their content.

Amongst other types, the types of meta data captured might be:

"Events": these define the actions or behaviours of an object acting upon another object or some other entity. The

event has three principal components: the key of the object performing the act (the "Actor"), the act being performed (the "Event Type"), and the key of the object or identity of an other entity upon which the act is being performed (the "Victim"). While simple, this structure allows a limitless series of behaviours and relationships to be defined. Examples of the three components of an event might be:

Actor	Event Type	Victim
Object 1	Creates Program	Object 2
Object 1	Sends data	IP Address 3
Object 1	Deletes Program	Object 4
Object 1	Executes	Object 2
Object 2	Creates registry key	Object4

"Identities": these define the attributes of an object. They include items such as the file's name, its physical location on the disk or in memory, its logical location on the disk within the file system (its path), the file's header details which include when the file was created, when it was last accessed, when it was last modified, the information stored as the vendor, the product it is part of and the version number of the file and it contents, its original file name, and its file size.

"Genesisactor"—the key of an object that is not the direct Actor of an event but which is the ultimate parent of the event being performed. For example in the case of a software installation, this would be the key of the object that the user or system first executed and that initiated the software installation process, e.g. Setup.exe.

"Ancillary data": many events may require ancillary data, for example an event such as that used to record the creation of a registry run key. In this situation the "event" would identify the Actor object creating the registry run key, the event type itself (e.g. "regrunkey"), and the Victim or subject of the registry run key. The ancillary data in this case would define the run key entry itself; the Hive, Key name and Value.

"Event Checksums": because the event data can be quite large extending to several hundred bytes of information for a single event, its identities for the Actor and Victim and any ancillary data, the system allows for this data itself to be summarised by the Event Checksums. Two event checksums are used utilising a variety of algorithms, such as CRC and Adler. The checksums are of the core data for an event. This allows the remote computer **2** to send the checksums of the data to the central computer **3** which may already have the data relating to those checksums stored. In this case, it does not require further information from the remote computer **2**. Only if the central computer **3** has never received the checksums will it request the associated data from the remote computer **2**. This affords a considerable improvement in performance for both the remote and central computers **2**,**3** allowing much more effective scaling.

Thus, the meta data derived from the remote computers **2** can be used at the community database **7** to define the behaviour of a process across the community. As mentioned, the data may include at least one of the elements mentioned above (file size, location, etc.) or two or three or four or five or six or all seven (or more elements not specifically mentioned here). This may be used accordingly to model, test and create new automated rules for use in the community database **7** and as rules that may be added to those held and used in the local database of the remote computers **2** to identify and determine the response of the remote computers **2** to new or unknown processes and process activity.

Moreover, it is possible to monitor a process along with any optional sub-processes as an homogenous entity and then compare the activities of the top level process throughout the community and deduce that certain, potentially malevolent practices only occur when one or more specific sub-processes are also loaded. This allows effective monitoring (without unnecessary blocking) of programs, such as Internet Explorer or other browsers, whose functionality may be easily altered by downloadable optional code that users acquire from the Internet, which is of course the principal source of malevolent code today.

The potentially high volume of active users gives a high probability of at least one of them being infected by new malware. The speed of propagation can be detected and recorded so that the propagation of malware can be detected and malware designated as bad on the basis of the speed of propagation, optionally in combination with the other factors discussed above, such as file size, location and name. The simple volume of infection can also be used as a trigger. In a further embodiment, difference of naming of an otherwise identical piece of code combined with acceleration of first attempts to execute the code within the community allows pattern matching that will show up an otherwise identically signatured piece of code as bad.

This feature allows the statement in some embodiments that "nothing will propagate in our community faster than X without being locked down", so that if any process or event propagates more quickly over a given duration, it is marked as bad. This is for reasons of safety given that if for example an object is propagating quickly enough, then it might infect computers before it can be analysed to determine whether or not it is malware.

This process can be automated by the identification of the vector of propagation in the community (i.e. the source of type of propagation), from timestamp data held in the community database and the marking of a piece of code that has these attributes as bad. By comparison, it is believed that all other anti-malware providers rely on a simplistic known bad model and therefore are reliant primarily on malware infection actually occurring on terminals and being reported.

Thus, the community database **7** can be used to make early diagnosis, or simply to take precautionary measures, and thus stop potentially fast propagating worms and other malware very, very early in their life cycle. Given that it is possible to create a worm that can infect every computer connected to the Internet within a matter of a few minutes, this feature is highly desirable.

Even faster determination may be made by combining data defining the speed of propagation of a new piece of software with metadata collected by the agent software from the remote computers **2** and fed to the community database **7**. This includes monitoring processes that attempt to conceal themselves from the user by randomly changing name and/or location on the remote computers **2**. It also includes a process's attempt to create an identical copy (i.e. with identical code contents) on the computer but with a different name. This is a classic attribute of a worm.

The signature of an object may comprise or be associated with a mask which can be built up with use of that object and which indicates the particular types of behaviour to be expected from the object. If an object is allowed to run on a remote computer **2**, even if the initial signature search **22** indicates that the object is safe, then operation of that object may be monitored within the parameters of the mask. The mask might indicate for example, the expected behaviour of the object; any external requests or Internet connections that that object might legitimately have to make or call upon the remote computer **2** to make, including details of any ports or interfaces that might be required to be opened to allow such

communication; any databases, either local or over a local area network or wide area network or Internet, that may be expected to be interrogated by that object; and so on. Thus, the mask can give an overall picture of the expected "normal" behaviour of that object.

In practice, therefore, in one embodiment the behaviour of the object is continually monitored at the remote computer(s) **2** and information relating to that object continually sent to and from the community database **7** to determine whether the object is running within its expected mask. Any behaviour that extends beyond the mask is identified and can be used to continually assess whether the object continues to be safe or not. Thus, if for example the object, on a regular basis (say monthly or yearly) opens a new port to update itself or to obtain regular data, then this information is flagged. If it is found that the object has done this on other remote computers and has had no ill effects, or this behaviour is known from other objects and known to be safe, then this behaviour might be considered as safe behaviour and the mask is then modified to allow for this. If it has been found previously that this new behaviour in fact causes unsafe or malevolent results, then the object can then be marked as unsafe even if previously it was considered safe. Similarly, if the object attempts to connect to a known unsafe website, database or to take action that is known as generally being action only taken by unsafe programs, then again the object may be considered to be unsafe.

This is shown schematically in FIG. **3**. FIG. **3** also shows the concept that any object can be pre-authorised by, for example a trusted partner, such as a major software company, a verification authority, a Government department, and so on. Pre-authorisation enables a supplier of a new object, which has not been released before, to get pre-authorisation for that object, and optionally includes the provision by that supplier of a mask detailing the expected and allowable behaviour of that object.

Referring to FIG. **3** for example, when a process is run, the local and/or community databases are searched as before at step **31**. If the process is not a pre-authorised one, then the steps of FIG. **2** may be taken and the process might be allowed to run or not at step **32**. If the process is pre-authorised, as determined at step **33**, then it is immediately allowed to run, step **34**. This may terminate the operation of the method. However, in a preferred variation, the process is then monitored whilst running, and is monitored each time it is run in the future in a monitoring state step **35** to determine whether its behaviour falls within its pre-authorised mask **36**. If the behaviour falls within the pre-authorised behaviour, then the process is allowed to continue to run. If the behaviour extends beyond the allowed mask, such as by trying to instigate further processes or connections that have not been pre-authorised, then this behaviour is flagged at an alert step **37**. Various actions could be taken at this stage. The process might simply not be allowed to run. Alternatively, the trusted authority that initially enabled pre-authorisation might be contacted, who may be able to confirm that this behaviour is acceptable or not. If it is acceptable, then the mask could be modified accordingly. If not acceptable, then the process might be marked as unsafe. Many other actions may be taken upon the noting of such an alert state.

If the process has been found not to be pre-authorised at step **33** but is nevertheless allowed to run, then the process is monitored at step **38** in order to generate a mask **39** representative of the normal behaviour of that process. Data representative of this mask might be sent to the community database **7** for scanning when other computers run that process. By continually monitoring a process each time it is run or during running of the process, any behaviour that differs from previous behaviour of the process can be noted and the mask can be modified, or this behaviour might be used to determine that a process that was once considered safe should now be considered to be unsafe.

In another embodiment, a computer **2** may have agent software installed that periodically or on-demand provides information to the community database **7** that is representative of all or selected ones of the software products loaded on or available the computer **2**. In particular, this may be information on one or more of: all the locally-loaded security products (such as anti-malware systems including anti-virus software, anti-spyware, anti-adware and so on), firewall products, specific settings and details of which signature files are currently loaded, version details for the operating system and other software, and also information such as which files are operating and the particular version and software settings at any time. (It will be understood from the following that auditing and testing for a match for more of these criteria increases the likelihood of computers being very similarly arranged and thus reduces the rate of false negatives and positives during the match search.)

The information relating to these software products, etc. may be provided individually to the community database. Preferably however, again for reasons of data quantity for storage and transmission, the information is provided as a signature or key representative of the information (e.g. by a hashing or compression function at the computer **2**). FIG. **4** shows schematically how the details of all local security products, versions, signature files, firewall settings, etc. **40** are used to create a key **41**. The key is transmitted to the community database **7**.

Since the community database **7** is provided with such information from many, possibly millions, of users' computers **2**, it is likely to hold corresponding information for other computers **2** that have the same or a similar configuration of security products, etc. Thus, the community database **7** can be searched at step **42** for other computers **2** having the same or a similar combination of security products including the same setting, signature files loaded and so on.

The community database **7** in this embodiment is also provided by the agent software with details of processes run by every computer **2** and thus knows whether or not a process has been detected by each computer **2**.

In this way, the community database **7** can be used to obtain information as to whether for example a particular, specific combination of operating system and various security products, including settings and signature files existing at a point in time, renders a particular computer **2** having those products and settings susceptible or vulnerable to any particular malware object.

In a simple example, if for example the database knows that a computer in the past has version A of anti-virus product B with downloaded signature update C, and also has a firewall D with particular settings E, and perhaps anti-spyware software F with signature updates G, but that a particular malware process P was not detected by this combination of programs at that point in time, then this information can be provided to a computer **2** that is known as having that combination of security programs/settings, and can be used to indicate that that computer **2** is vulnerable in the short term to attack by that particular malware process. This information can be presented to the user either by displaying a window **43** on the screen display of the computer **2** or by directing the user to a particular website which explains the position in more detail. The user might be informed for example that their particular combination of security products, etc., exposes their computer to a risk of being infected by the Sobig virus as that virus

is not detectable by their computer. The user might be offered specific advice (e.g. to update a particular anti-virus program with a particular signature file) or software to download and install to remove the risk.

Thus, the community database **7**, when provided with information relating to all the security products, etc. on a particular computer at a particular time, is searched for events for processes marked as "bad" that occurred on computers with that particular mix of security products and that were not locally detected. This information can then be fed back to the user of the particular computer, for example directly or by directing the user to a website. This information can be provided virtually in real-time, allowing a new user or a user of a new computer to be able to increase the computer's effective security very quickly.

The preferred method also tracks which objects are related to each other and uses the concept of ancestry to enable objects to be marked as malware. For example, any particular process may spawn child processes which are therefore related. The key relating to the first object may be inspected to identify a component that uniquely identifies the first object and that is inherited or otherwise present in the key of a descendant or other related object of the first object. This component is referred to herein as a "gene". This general technique may be used in a number of ways:

a) A known and trusted parent process is afforded the ability to create child processes which may be automatically marked as safe to run on the local computer. It is also possible that this "inherited" property may be passed down to grand children processes and so on. This safe status is passed to the parent's child processes and possibly, through them, further child processes (referred to here as "issue"), such signatures for the issue can all automatically be recorded in the local database as good. This allows the issue processes to be quickly marked as good, even if a connection to the community database **7** is not available.

b) By monitoring activity of the parent process, if it is later found that the parent process is malware, then all of the issue processes can all automatically be recorded in the local database as bad.

C) Similarly, by monitoring activity of the issue processes, if it is later found that one of the issue processes is malware, then one or more of the parent process and all of the other issue processes (i.e. all of the related processes in this context) can all automatically be recorded in the local database as bad.

d) Parental creation of a signature for a child or children including the ability for these to be automatically marked as either good or bad depending on the parent's behaviour and determination. Note that in some embodiments the product can "watch" or monitor the birth of a child process and automatically create the signature upon arrival. This provides the ability to monitor the creation of a bad program by another bad program. It is possible therefore to monitor the ancestry of a program so if for example the grandfather creates a program (the father) and this in turn creates a bad program (the son), it is possible automatically to determine the father as a bad program.

e) A feature may be included that allows for automatic forfeiture of a child's inherited ability to trigger the automatic creation of signatures on any further births because the child, as parent, has already produced bad offspring. Preferably, a rule is that if a file has one bad offspring then the inherited ability can be automatically removed.

f) An ability to watch clones or identical twins of objects (e.g. the same process running on other systems in the community) to compare the pattern of their issue and to make decisions as to whether or not to treat any particular process as malware.

One or more of these features a) to f) can be used to provide a solution to the problem of producing a security product that can be used effectively without 100% reliance on being permanently connected to the Internet, which is often impractical. Examples of this are Windows Update and other processes used more and more by vendors who wish to be able to roll out product updates automatically across the worldwide web.

Possible benefits of these types of features above conventional software are as follows. Antivirus software tends to have a cache of known bad signatures. The problem is keeping this up to date. Take the simple example of someone buying a new computer. The computer comes with an antivirus product preloaded with a signature cache. Between the time when the PC was built, shipped to the store and bought by the user several days or weeks will have passed. The user starts the PC and is exposed to any new virus or malware which was created after the PC was built. Full protection requires the user to connect to the internet and download updates. This cannot be guaranteed to occur ahead of other activities by the user on the internet (almost physically impossible to guarantee). With a local cache of known good processes, as in the embodiments of the present invention, it is possible to ship the computer/terminal preloaded with a pregenerated cache of signatures for all of the good (clean) software preloaded by the computer manufacturer. In this case the user can connect to the internet knowing that any new or updated programs will be immediately detected and verified. Also any auto-updating software can function forcing signatures to be automatically built for its children and more remote off-spring (i.e. grandchildren, great-grandchildren, etc).

Reference is now made to "Defeating. Polymorphism: Beyond Emulation" by Adrian Stepan of Microsoft Corporation published in "Virus Bulletin Conference October 2005" and also to U.S. 60/789,156 filed on 5 Apr. 2006, the entire content of which are hereby incorporated by reference. In that paper and patent application, there are disclosed methods of decrypting files to allow the files to be analysed to determine whether or not the file actually is malware. In U.S. 60/789,156 in particular, there is disclosed a method of unpacking/decrypting an executable computer file using a host computer, the method comprising: partitioning the executable computer file into plural basic blocks of code; translating at least some of the basic blocks of code into translated basic blocks of code that can be executed by the host computer; linking at least some of the translated basic blocks of code in memory of the host computer; and, executing at least some of the translated basic blocks of code on the host computer so as to enable the executable computer file to be unpacked or decrypted, whereupon the unpacked or decrypted executable computer file can be analyzed to determine whether the executable computer file is or should be classed as malware. There is also disclosed in U.S. 60/789,156 a method of unpacking/decrypting an executable computer file, the method comprising: partitioning the executable computer file into plural basic blocks of code; creating at least a read page of cache memory for at least some of the basic blocks, the read page of cache memory storing a read cached real address corresponding to a read cached virtual memory address for the respective basic block, and creating at least a write page of cache memory for at least some of the basic blocks, the write page of cache memory storing a write cached real address corresponding to a write cached virtual memory address for the respective basic block;

and, emulating the executable file by executing at least some of the basic blocks of code so as to enable the executable computer file to be unpacked or decrypted, whereupon the unpacked or decrypted executable computer file can be analyzed to determine whether the executable computer file is or should be classed as malware; wherein during the execution of a basic block, at least one of the read page and the write page of cache memory is checked for a cached real address corresponding to the virtual address that is being accessed for said basic block.

The techniques disclosed in these documents can be used in the present context when it is desired to analyse a file in detail. More generally however, the techniques disclosed in these papers, and particularly the enhanced techniques disclosed in U.S. 60/789156, can be used to provide information about the activity of a file when it is run on a computer because the techniques disclosed in these papers emulate the running of the file and therefore allow the file's activity to be interpreted.

A further situation arises when users wish to install software while offline. In a preferred embodiment, when a user attempts to install new software while offline, agent software running on the user's computer **2** prompts the user for authorisation to allow the installation process to proceed such that the execution of the installation can itself be "authorised" by the user. This authorisation by the user is treated as a "Genesis" event and will be so termed hereinafter. However there are some processes commonly used in installation of software that communicate with other existing programs on the installation machine, e.g. Microsoft's MSIEXEC.

The Genesis approach involves a process that generates signatures as a result of the user's authorisation on the user's computer **2**. Those signatures are stored in the local database on the user's computer **2**. In one embodiment, those locally stored signatures are referred to as necessary by the agent software during the installation process so that the installation can proceed. Alternatively, the security checks made by the agent software can be switched off during the installation process. The switching off may only be for a limited duration, such as a few minutes which should be sufficient to allow most software to be installed, the off time optionally being user-configurable.

In any event, once the installation has been completed and the user's computer **2** connected to the community database **7**, the agent software on the user's computer **2** can upload the signatures relating to the installation from the user's local database to the community database **7**. With corresponding data from other users' computers **2**, the community database **7** can then be used to make a rapid determination that the installation of this particular software is benign.

As a variant to this embodiment, when a user's computer **2** is at some point in time on-line, the agent software on the user's computer **2** may download signatures of a so-called "trusted installer" or "licensed installer". This allows the operation of a method such that a "licensed installer" and any child processes of the licensed installer are permitted to execute while a Genesis event is "current", e.g. within a period of minutes after an authorisation from the user. Preferably signatures of licensed installers are always downloaded, as and when added to the community database **7**, to a remote computer **2** while online.

There may be further refinements to this method, such as to prevent licensed installers executing if a media change has occurred during the currency of a Genesis event. However, any unknown processes, which may be malware, may still be detected and blocked. Having a small number of licensed installers facilitates download, as minimal data is required, especially compared to downloading virus signature files. As another example, super-trusted installers, such as for example "Windows Update", may be employed whereby all new processes created by the super-trusted installer are marked immediately as safe.

In the case of an installation occurring when the remote computer **2** is connected to the community database **7**, another option is that if the software that is about to be installed is not known to the community database **7**, then the system will block the installation or alert the user. For example, a message may be displayed at the user's computer **2** to the effect that "You are about to install some software. This software is not known to [the community]. Are you sure you wish to proceed?".

Embodiments of the present invention have been described with particular reference to the examples illustrated. However, it will be appreciated that variations and modifications may be made to the examples described within the scope of the present invention.

The invention claimed is:

1. A method comprising:

receiving, at a base computer, details uniquely identifying one or more security products operating at a point in time on a remote computer;

receiving, at the base computer, details uniquely identifying one or more security products operating on other remote computers in communication with the base computer;

receiving, at the base computer, details of a process that has been executed by at least one of the other remote computers;

determining, by the base computer and based on the received details of the process that has been executed by the least one of the other remote computers, that the process is a malware process not identified by the one or more security products operating on the at least one of the other remote computers; and

determining, by the base computer, that the remote computer is vulnerable to the malware process, wherein the determination is based on the at least one of the other remote computers having a same or similar combination of security products as the combination of security products operating on the remote computer.

2. The method according to claim **1**, further comprising:

providing information to the user of the remote computer that the remote computer may be susceptible to attack by the malware processes.

3. The method according to claim **1**, wherein the details of the one or more security products includes the name of the security products, versions, and loaded signature files.

4. An apparatus comprising:

a base computer constructed and arranged to receive details uniquely identifying one or more security products operating at a point in time on a remote computer;

the base computer being constructed and arranged to receive details uniquely identifying one or more security products operating on other remote computers in communication with the base computer;

the base computer being constructed and arranged to receive details of a process that has been executed by at least one of the other remote computers;

the base computer being constructed and arranged to determine, based on the received details of the process that has been executed by the least one of the other remote computers, that the process is a malware process not identified by the one or more security products operating on the at least one of the other remote computers; and

the base computer being constructed and arranged to determine that the remote computer is vulnerable to the malware process, wherein the determination is based on the at least one of the other remote computers having the same or similar combination of security products as the combination of security products operating on the remote computers.

5. The apparatus according to claim **4**, wherein the base computer is constructed and arranged to provide information to the user of the remote computer that the remote computer may be susceptible to attack by the malware process.

6. The apparatus according to claim **4**, wherein the details of the one or more security products includes the name of the security products, versions, and loaded signature files.

7. A computer program recorded on a non-transitory computer readable medium comprising program instructions for causing a computer to perform a method according to claim **1**.

8. A computer program recorded on a non-transitory computer readable medium comprising program instructions for causing a computer to perform a method according to claim **2**.

9. A computer program recorded on a non-transitory computer readable medium comprising program instructions for causing a computer to perform a method according to claim **3**.

10. The method according to claim **3**, wherein the details of all or selected security products further includes settings at a particular point in time.

11. The method according to claim **1**, further comprising:

identifying, by the base computer, one or more of the other remote computers having the same or similar combination of security products as the combination of security products operating on the at least one of the other remote computers; and

determining, by the base computer, that the identified one or more of the other remote computers having the same or similar combination of security products as the combination of security products operating on the at least one of the other remote computers is vulnerable to the malware process.

12. The apparatus according to claim **6**, wherein the details of all or selected security products further includes settings at a point in time.

13. A method comprising:

receiving, at a base computer, details uniquely identifying one or more security products operating at a point in time on a remote computer;

receiving, at the base computer, details uniquely identifying one or more security products operating on other remote computers in communication with the base computer;

receiving, at the base computer, details of a process that has been executed by at least one of the other remote computers;

determining, by the base computer and based on the received details of the process that has been executed by the least one of the other remote computers, that the process is a malware process;

determining, by the base computer, that the remote computer is vulnerable to the malware process, wherein the determination is based on the at least one of the other remote computers having a same or similar combination of security products as the combination of security products operating on the remote computer;

identifying, by the base computer, one or more of the other remote computers having the same or similar combination of security products and settings as the combination of security products operating on the remote computer; and

identifying, by the base computer, one or more malware processes that were not identified by the combination of security products and settings operating on the one or more of the other remote computers having the same or similar combination of security products as the combination of security products operating on the remote computer.

14. The method according to claim **1**, wherein the one or more security products include a firewall product and an antivirus product.

15. The apparatus according to claim **4**, wherein the one or more security products include a firewall product and an antivirus product.

16. A method comprising:

receiving, at a base computer, details of one or more security products operating at a point in time on a remote computer;

receiving, at the base computer, details of one or more security products operating on other remote computers in communication with the base computer;

receiving, at the base computer, details of a process that has been executed by at least one of the other remote computers;

determining, by the base computer and based on the received details of the process that has been executed by the least one of the other remote computers, that the process is a malware process;

determining, by the base computer, that the remote computer is vulnerable to the malware process, wherein the determination is based on the at least one of the other remote computers having a same or similar combination of security products as the combination of security products operating on the remote computer;

identifying, by the base computer, one or more of the other remote computers having the same or similar combination of security products and settings as the combination of security products operating on the remote computer; and

identifying, by the base computer, one or more malware processes that were not identified by the combination of security products and settings operating on the one or more of the other remote computers having the same or similar combination of security products as the combination of security products operating on the remote computer.

* * * * *